Video Game Development

Ruth Alsobrook-Hurich
3/12/2021

NOTE:
All demos are on a Windows Based Computer. If you have a MAC Operating System, there may be a few variances. Be sure to follow your MAC rules and regulations.

If you download any folders from this course, they come **COMPRESSED**. *You must "Extract"* the files in order to use the content (unless otherwise stated). This is accomplished by *Right Clicking on the file > Extract Here*.

Each section of this document has a video at the end of the topic. You will find a URL link; clicked upon the link to view the video. Internet connection needed to view the videos.

There are also a few links to other sites, sounds, and craziness. These links are usually underlined, and in a different color. Use *Ctrl + Click* on these to open; or, right click and copy the URL (the address of a World Wide Web page, another URL, and ANOTHER!).

Introduction

The Author

Hello. My name is Ruth Alsobrook-Hurich, and I will be your guide through this fantastic world of Video Game Development using the Unity application.

I have been working with Unity for several versions now. This started back in 2012 when I saw my son creating a game with the C# language, while using an **Integrated Development Environment** (IDE). To see a visual result of the game come from words astounded me. I had to know more. I had to learn more. I had to do this.

On that day, I signed up for a Video Game Development class. The creativity of it all was what really pulled me in. One could get lost in just this one aspect of Video Game Development. As you can see, I have stayed with it for a few years (and a few versions).

My hopes are to show you the very basics of the application, while creating a fun game. This specific course will not dive too deep into the “programming” aspect. Yes. We will use scripts; however, they will be very simple, and supplied to you.

Be prepared to be amazed at how powerful this small application can be.
Take your time in reading and doing the work.
Make “The Game” yours wherever possible.
Most of all, have fun.

Sister and Myself in Ireland 2015

Application

We will be using the **UNITY Game Engine** for this course. In fact, the install will come very shortly after all these words to you.

Unity is a “cross-platform game engine developed by Unity Technologies, which is primarily used to develop both three-dimensional and two-dimensional video games and simulations for computers, consoles, and mobile devices”. (Wikipedia)

This textbook will use **2020.2**, which is the latest at the time of this writing. I encourage you to install the latest version to follow along.

With that said; if you prefer to see things very similar, you can always download earlier versions of the Unity application by navigating to Unity’s Older Versions Download Archives. This can be found on the Download Archive page (https://unity3d.com/get-unity/download/archive?_ga=2.158099002.1060657556.1538601770-186294159.1503873147).

You will Instal these earlier versions, if wanted, within the Unity HUB. A simple process. If you wish to go this route, and have issues, contact me.

As stated, it is highly recommended for you to install the latest version.
Do not update until the course is completed.
There will be a chapter on installation. No need to go there now.

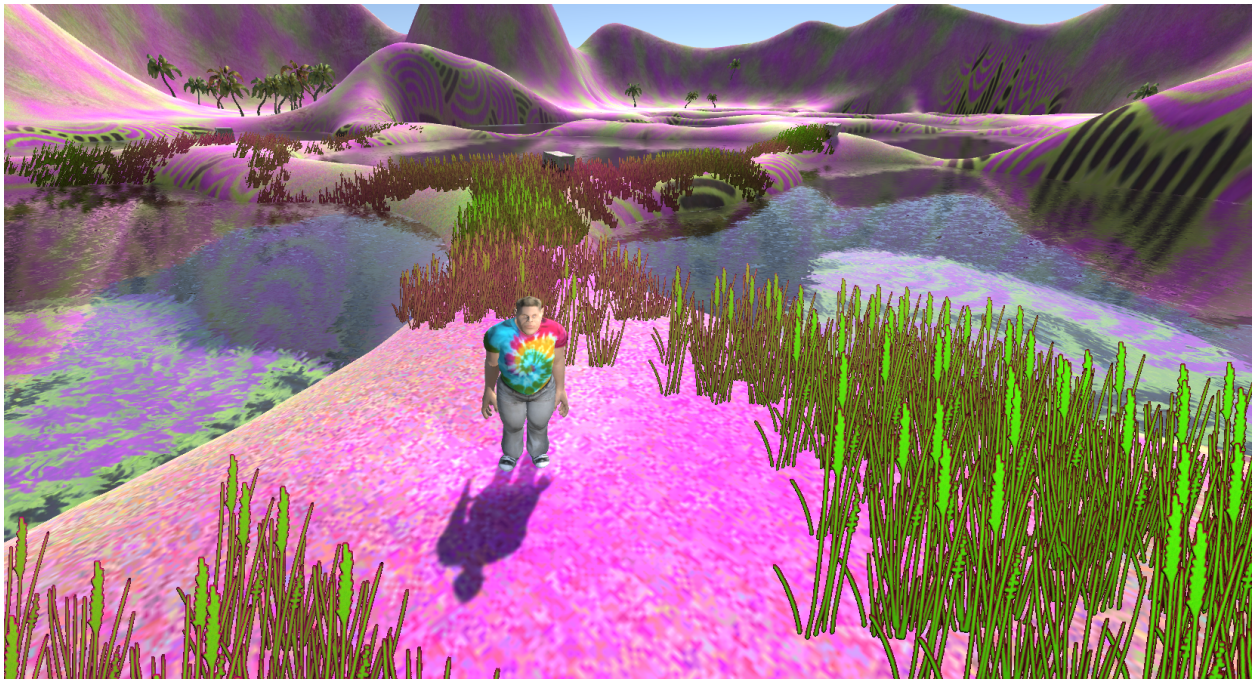
The Game

For this class, we will be creating a basic “Pick-Up” game. The game will start with an Intro Scene, which will have the game title, instructions, and an option to “play”.

Our player will run about an island, finding treasures to collect. Once all treasures are collected, a new scene appears stating that the game “has been won”.

In this basic “Pick-Up” game, we will add an Enemy. The Enemy will run with Artificial Intelligence about the island, trying to attack our Player. If the Enemy makes contact with the Player, a scene appears stating the game “has been lost”.

We will add an option to the Win/Lose Scenes to “Play Again” or “Quit”.



File Management for Course Work

Before we move any further, a folder needs to be created on your computer to save your Project within. This is a very important process. It is called **File Management**. This action will allow you to understand where your work is being saved, and how to find your project.

Unity normally selects its default location for Projects. This path looks like this C:\Users\Public\Documents\Unity Projects

We are going to circumvent this action to save files where we want them to go.
It is all about CONTROL of your File Management.

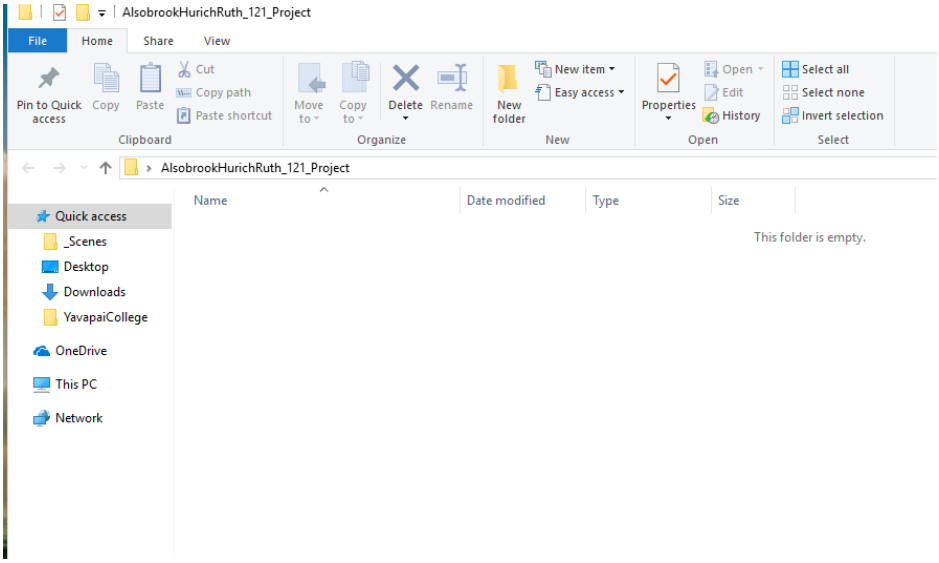
Choose your place to save projects. When you decide where this “place” is, in the blank area *Right Click > New > Folder*

Name your Folder LastFirst_121_Project (IE:SmithJohn_121_Project)

You should start using this MAIN FOLDER to house your work. If you get in this habit of creating a project folder first, and then associating it with Unity, you should have no problems finding your work.

File Management is one of the most frequent issues for new users with Unity.

EXAMPLE



https://www.youtube.com/watch?v=pTOOWoTITvc

NOTE:
This textbook will use **2020.2**, which is the latest at the time of this writing. I encourage you to install the latest version to follow along.

With that said; if you prefer to see things very similar, you can always download earlier versions of the Unity application by navigating to Unity's Older Versions Download Archives. This can be found on the Download Archive page (https://unity3d.com/get-unity/download/archive?_ga=2.158099002.1060657556.1538601770-186294159.1503873147).

You will Instal these earlier versions, if wanted, within the Unity HUB. A simple process. If you wish to go this route, and have issues, contact me.

As stated, it is highly recommended for you to install the latest version.
Do not update until the course is completed.
There will be a chapter on installation. No need to go there now.

Adobe Sparks

You will need some type of document to keep track of your progress, and the Assets used within the Project. An easy way to do this is with Adobe Sparks. This is a great way to show you work inside a well-known name brand product. It will also allow you to see your work after the course has been completed. Seeing the progression of your work is important for future employers.

The following video provides comedy and information on how to start with the Adobe Sparks.
This particular video was created for VGD 171.
Please be sure to associate VGD 121 with this Sparks Page.
Please find Adobe Sparks Privacy Policy → **HERE** ←
Please find Adobe Sparks Accessibility information → **HERE** ←

https://www.youtube.com/watch?v=B8PZeuqox-I
Example Page ⇒ **HERE** ⇐

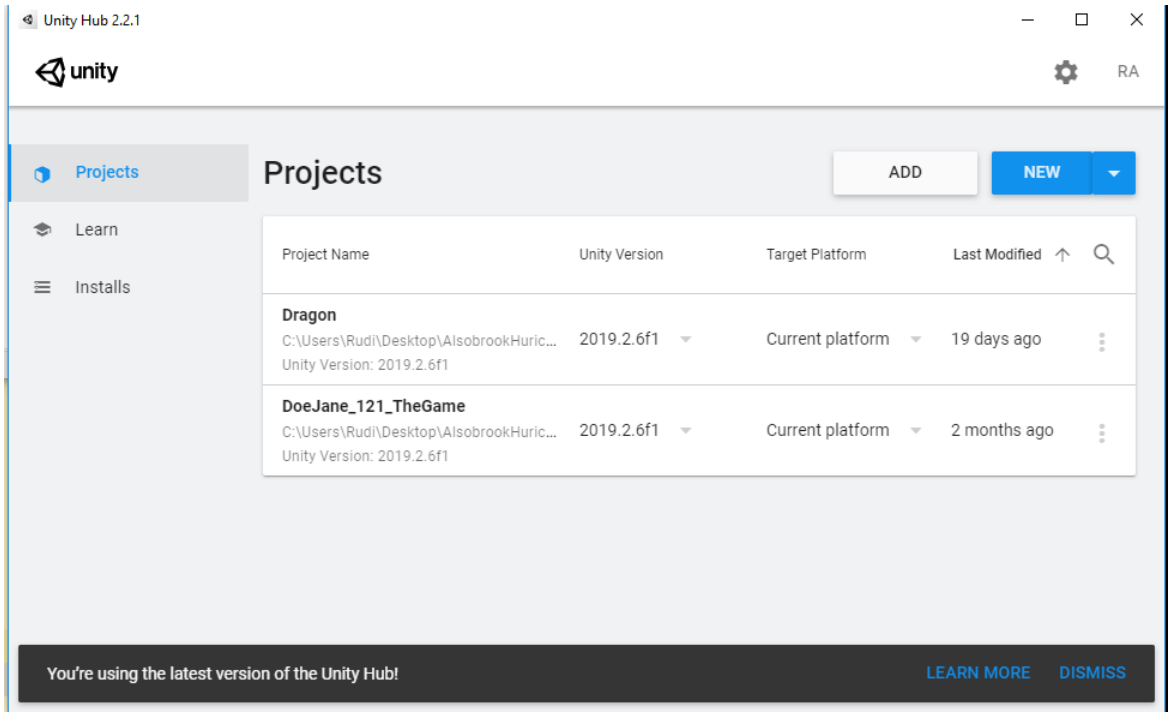
Unity Application Download

For this textbook we will be using the **2020.2** Unity 3D version.

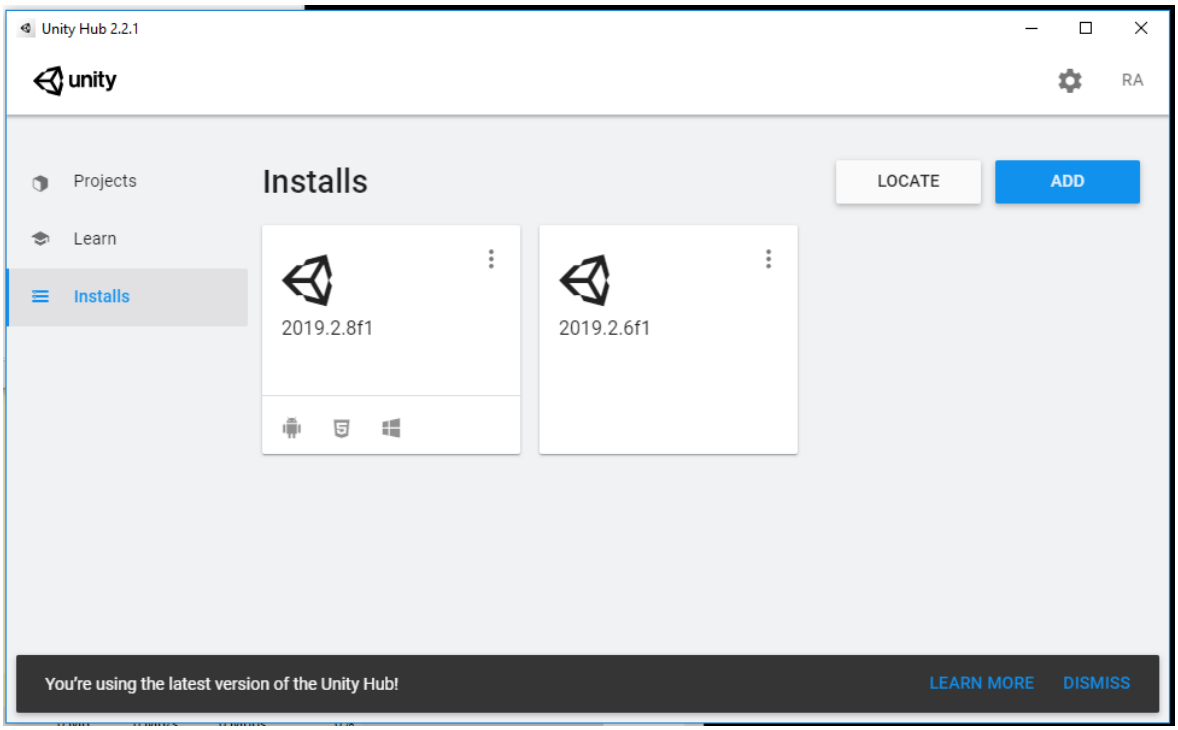
We need to install the Unity Hub to begin. Please go to
https://unity3d.com/get-unity/download
And select, **Download Unity Hub**

Double Click on the .exe file to Install the Hub. Follow the prompts.

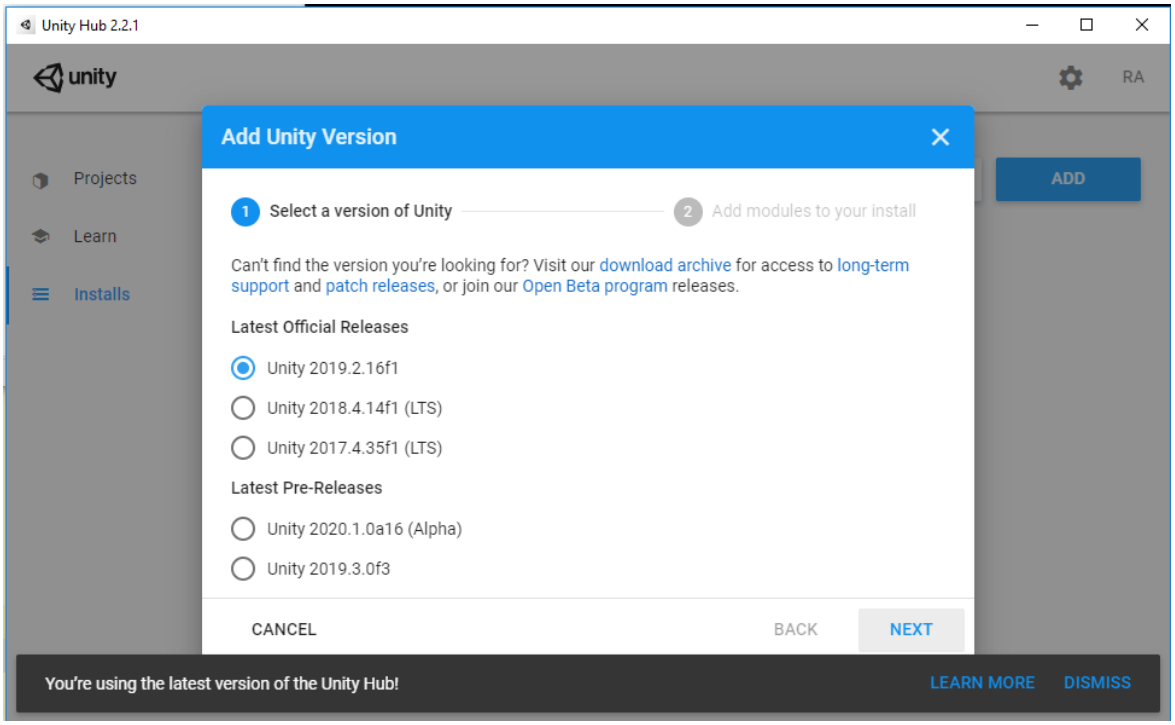
Once the Unity HUB has been installed, open this. A Dialog box will appear.



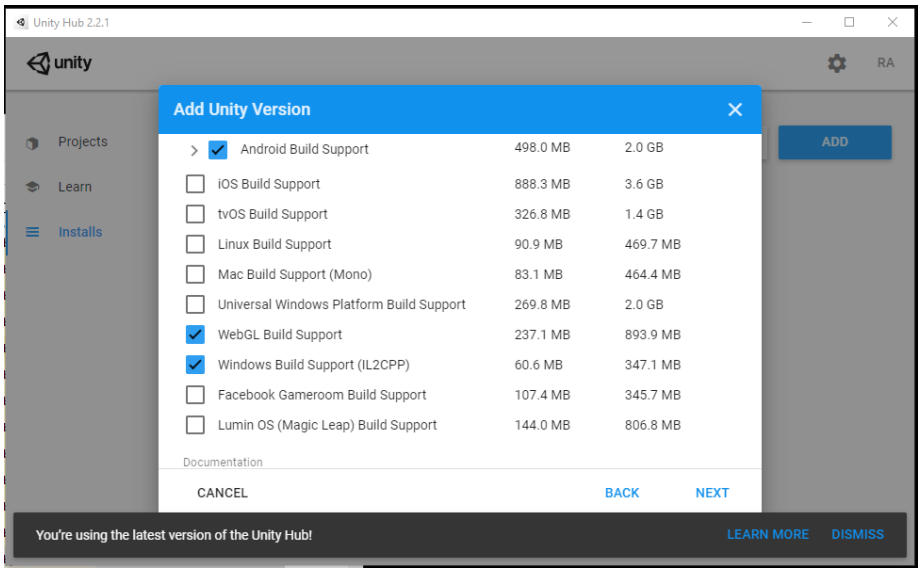
Select the **Installs** on the left hand side.



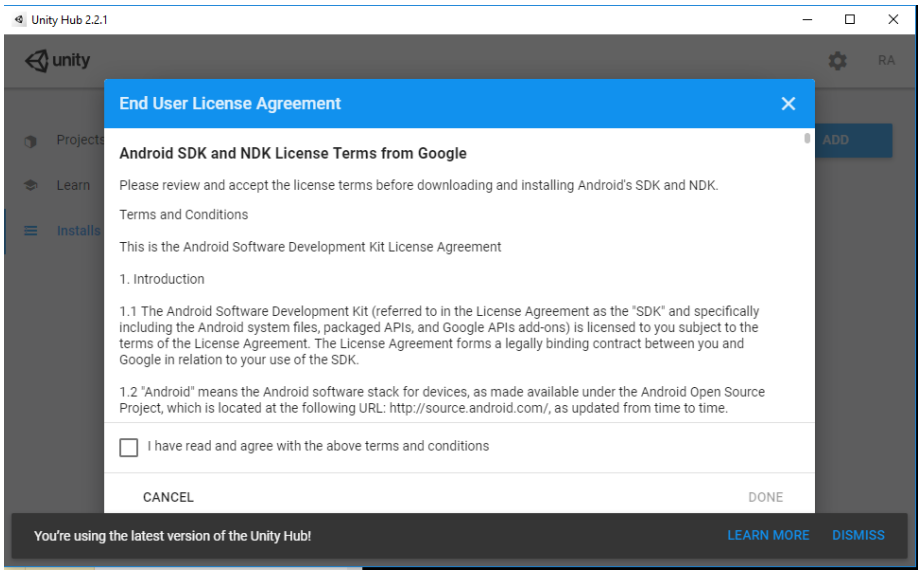
Select **Add**.



Select the Unity version to install and select Next. Select **Microsoft Visual Studio** (if not selected), **Android Build Support** (or IOS if on a MAC), **WebGL Build Support**, **Windows Build Support**, and **Documentation**. Then select Next.



Agree to the Terms and select Done.



Follow the install WYSIWYG (*what you see is what you get)* to install on your computer. ***You may need to also manually install the Microsoft Visual Studio. If the Unity install failed to bring Microsoft Visual Studio, navigate to https://visualstudio.microsoft.com/vs/unity-tools/, install the COMMUNITY version. It is FREE.***

https://youtu.be/8IngVb6wDPw
Video from BMo

https://youtu.be/DscSePG3myU
Another install video from myself while installing Unity Version 2019

IMPORTANT!
Be sure to install Microsoft Visual Studio. This is what is used for Programming (scripts). The application will provide the executable file for installation. In the past, this was automatic. Now, the IDE may need to be installed manually. If asked, you will use the COMMUNITY version. This is FREE.

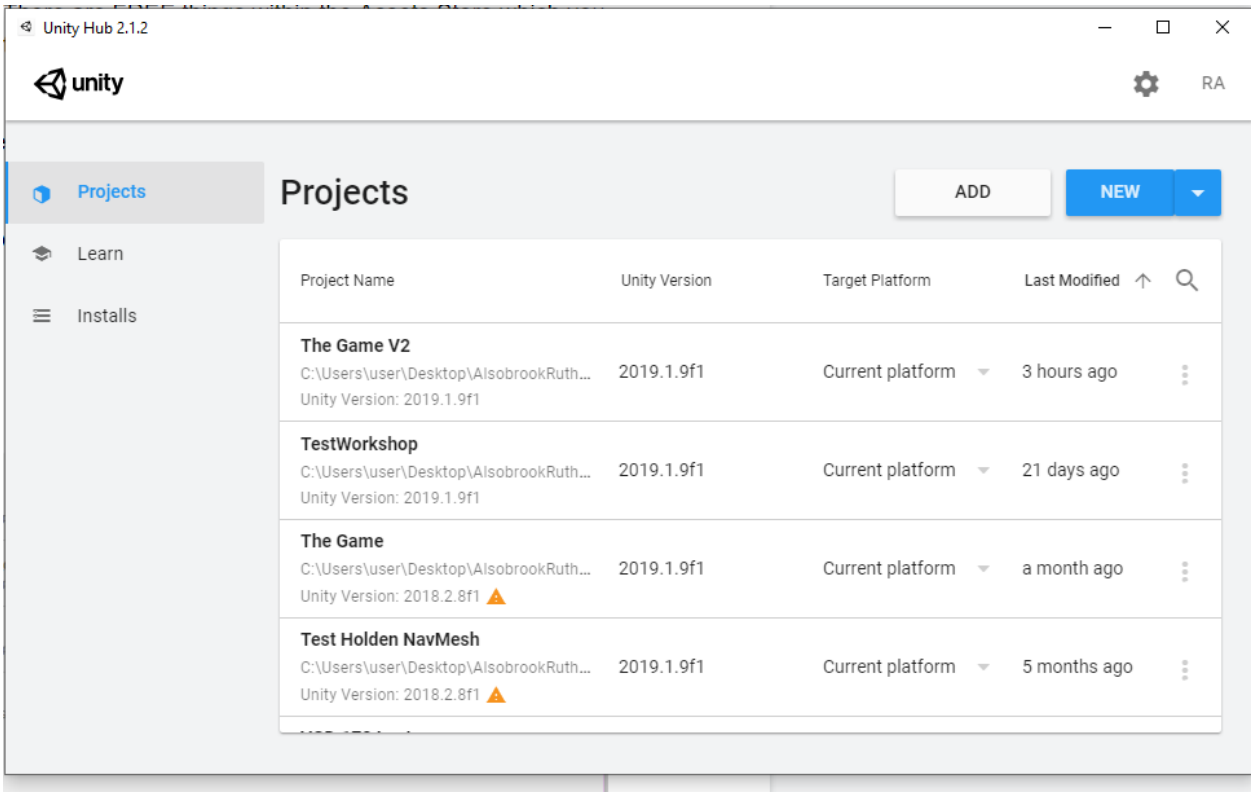
First Look

You need to create an account with Unity. This account allows you to utilize the vast Asset Store, and the cloud storage. The latter I have not used. For myself, I like the work on the computer used to create the project. This is just my preference. YOU need to find what works for you.

Remember your account information. There are FREE things within the Assets Store which you may wish to play around with later. With that said, know the more you add to the project, the larger the file, the slower the game.

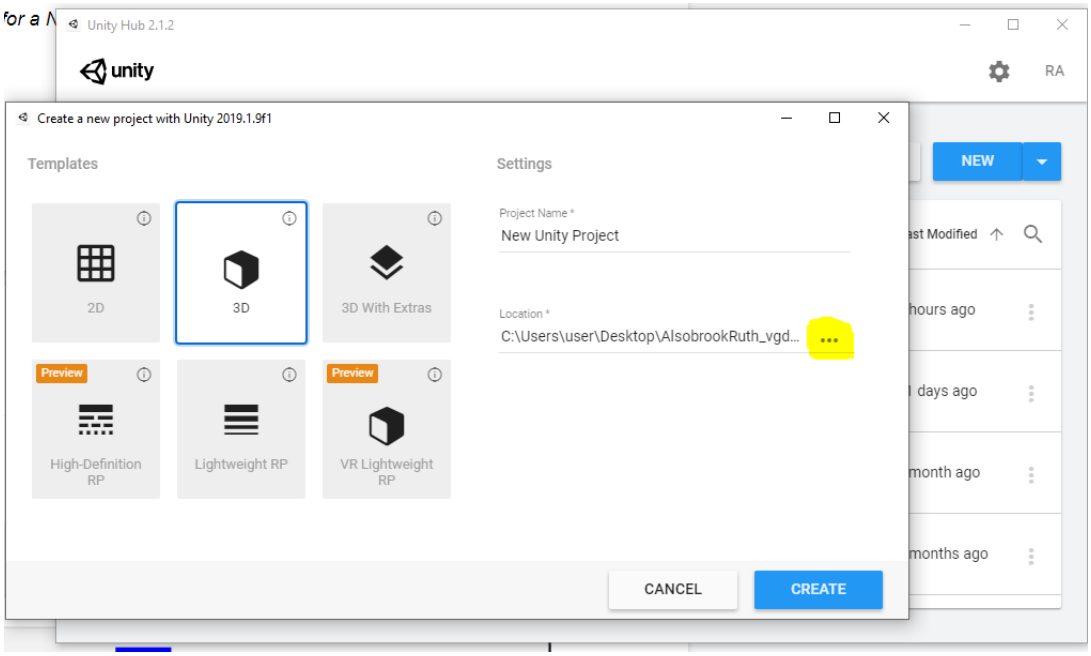
If you did not “Launch” Unity, open the Unity Hub and let us start looking at what is available..

When you first open the Unity Hub, you will see something similar to this:



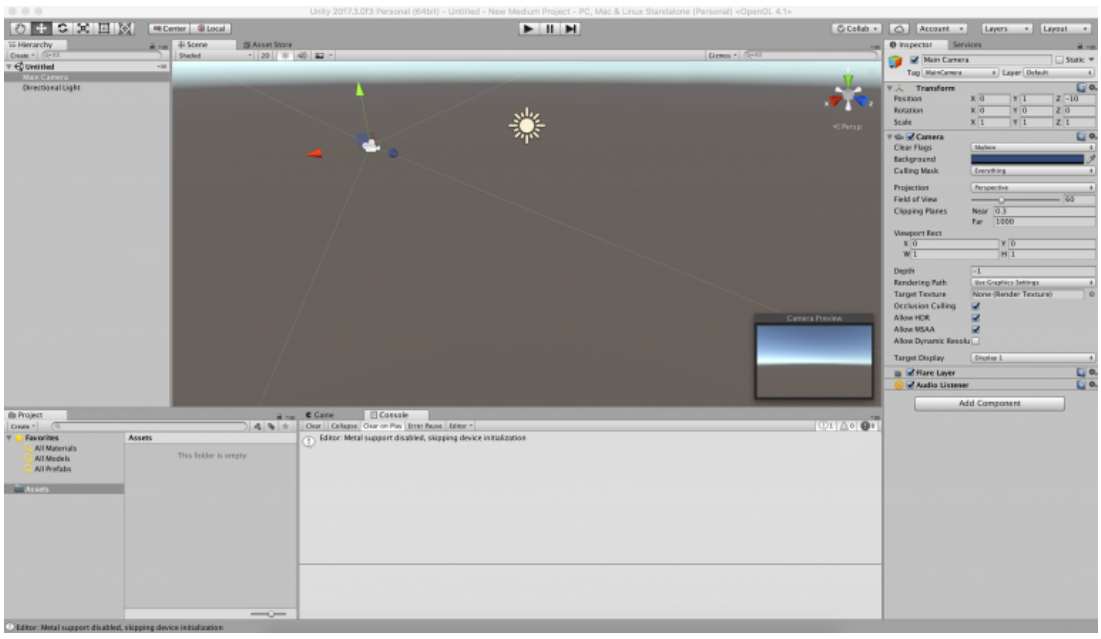
For our purposes, *Select “**NEW**” for a New Project*. Makes sense, yes?

This is what happens



Name your Project (SmithJohn_121_Projects). Select the “dots” (*Yellow Highlight*) to find your Project folder. Remember way back when you created that folder?

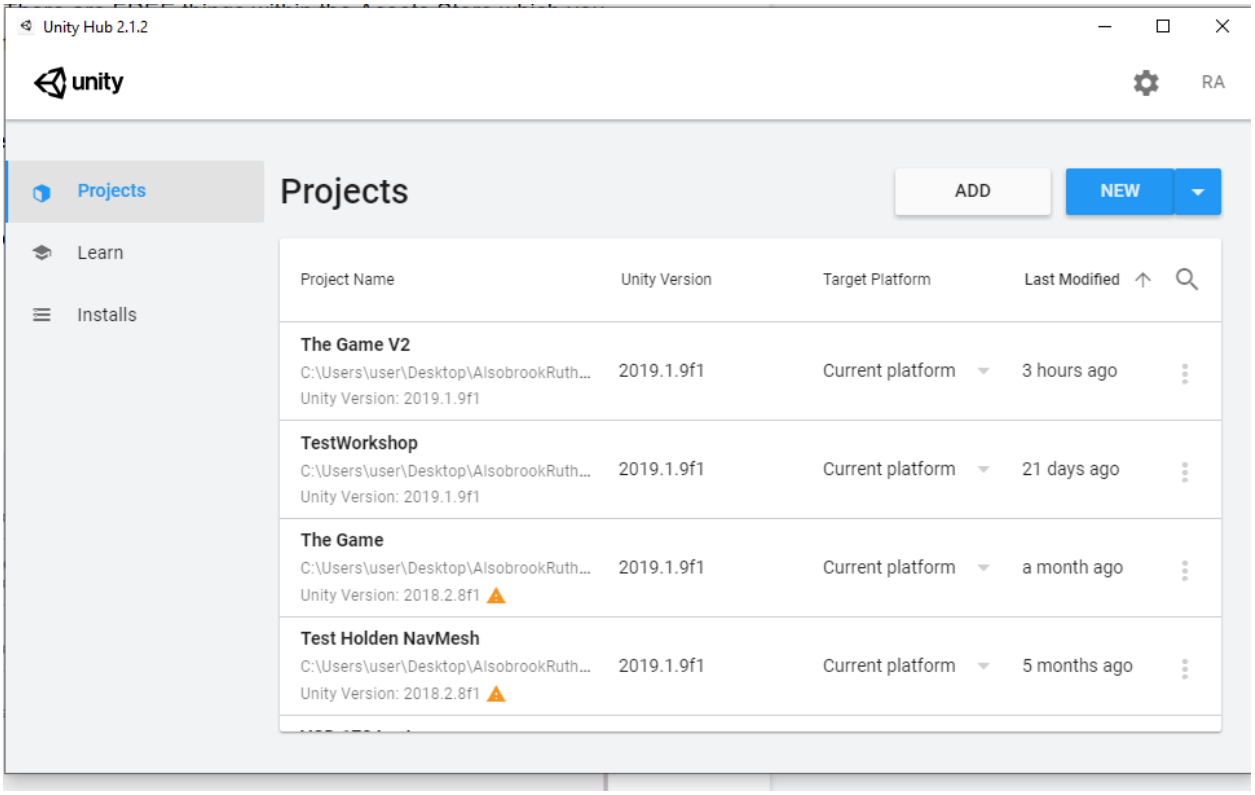
Make sure “3D” is selected. Click Create.
This is where the magic happens!
You should now see something, which looks similar to this



Let's explore what we have here. I am going to go straight to the horse's mouth, so to speak, and will use the Unity 3D Manual for this information. Open the following URL, and **Read** until you reach *Assets Workflow*.
http://docs.unity3d.com/Manual/UsingTheEditor.html

Be sure to save your project.
File > Save Project
How simple was that?! You have just installed, and opened, a magnificent program for game design!
Close Unity.

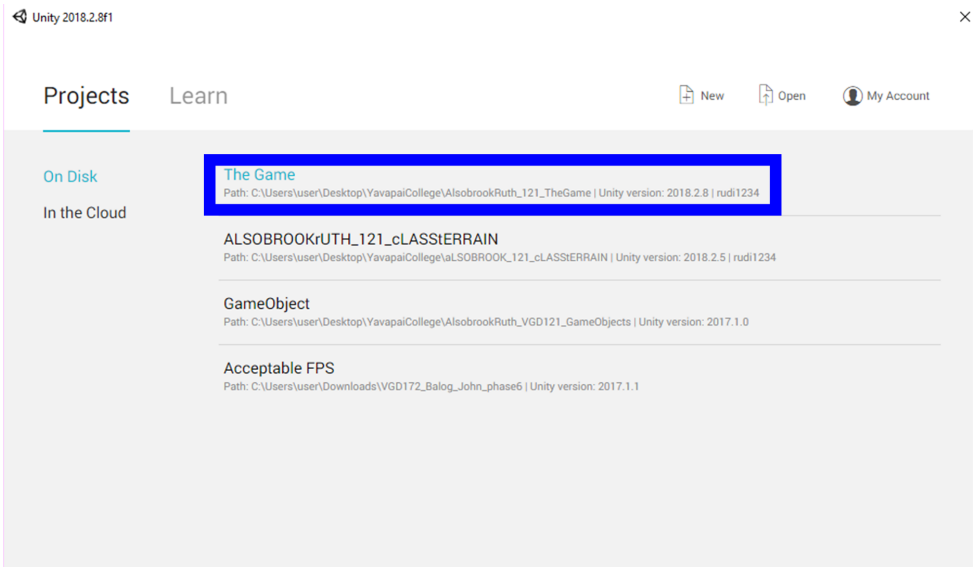
Remember, this project will show (after saving) in the HUB start window.



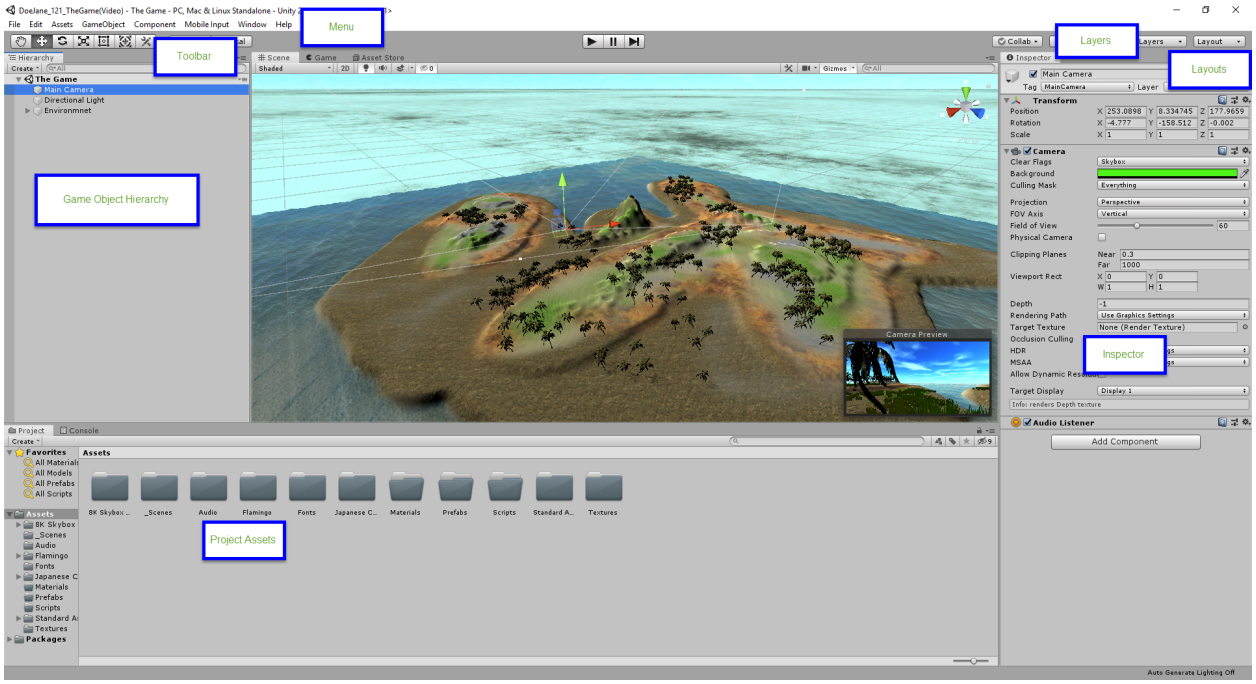
https://youtu.be/7Cg53mzdXbE
NOTE: In the video, I show you the process of importing the Standard Assets to your project. **PLEASE DO NOT DO THIS**. We will IMPORT the package later.

Unity Interface

Let's take a look at the **Unity Interface**.
Open Unity to your first project. If the project was closed prior to this reading, it will be the first project listed in the project dialogue box (Blue Box).

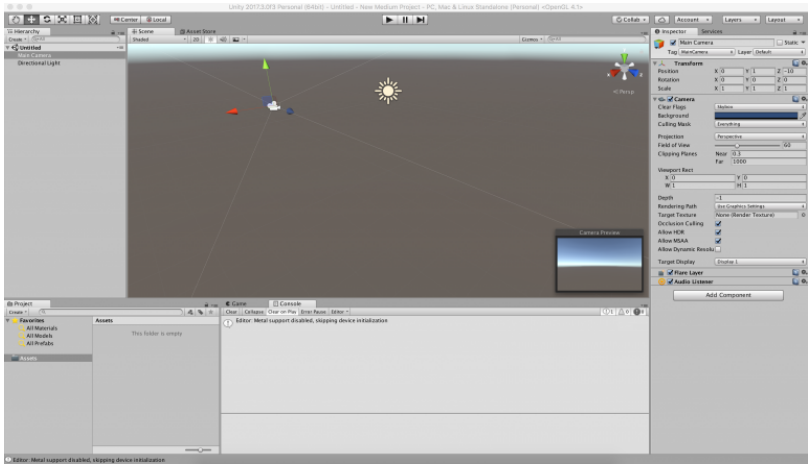


Double click the project to open it.

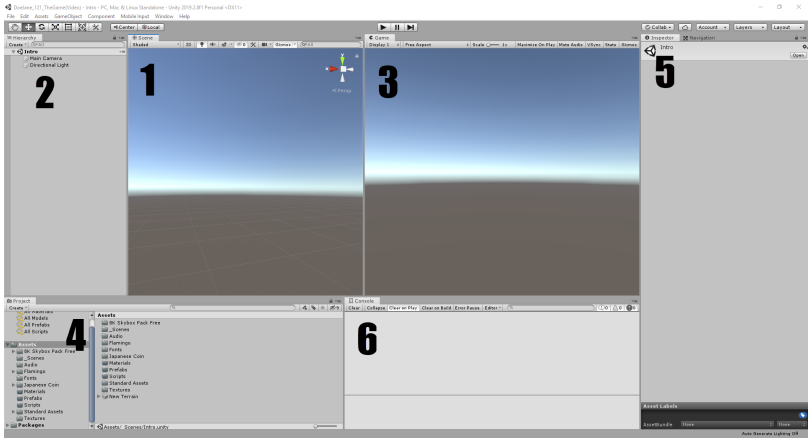


First and foremost, you, the user, can decide how to view your set-up. All open Windows can be moved by clicking and dragging. You can also use the Windows > Layouts option to change the view (Also under Layouts, right side of application).

Unity has created some layout defaults, which are commonly used, or you can make your own custom layout. With that said, please know that I do not have your custom layout and sometimes am unable to open projects saved with this custom look.



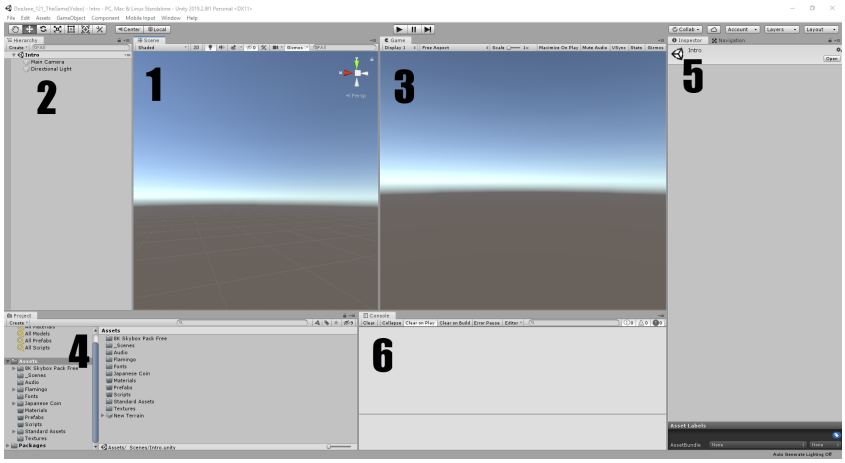
Second, we will look at the project Window. EVERYTHING in your project will fall under the main Assets folder (#4). This is the Window that holds all the components within the Project. It is a good habit to create folders for your work and keep everything organized. Let's try this right now.

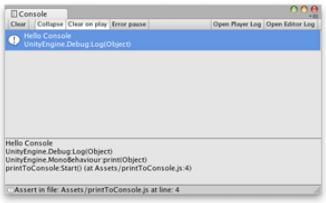


A Scenes Folder is created for you when you create a new Project. We'll be creating new folders for our project later. This is a simple process and is essential to File Management. Keeping everything organized will assist in so many ways!

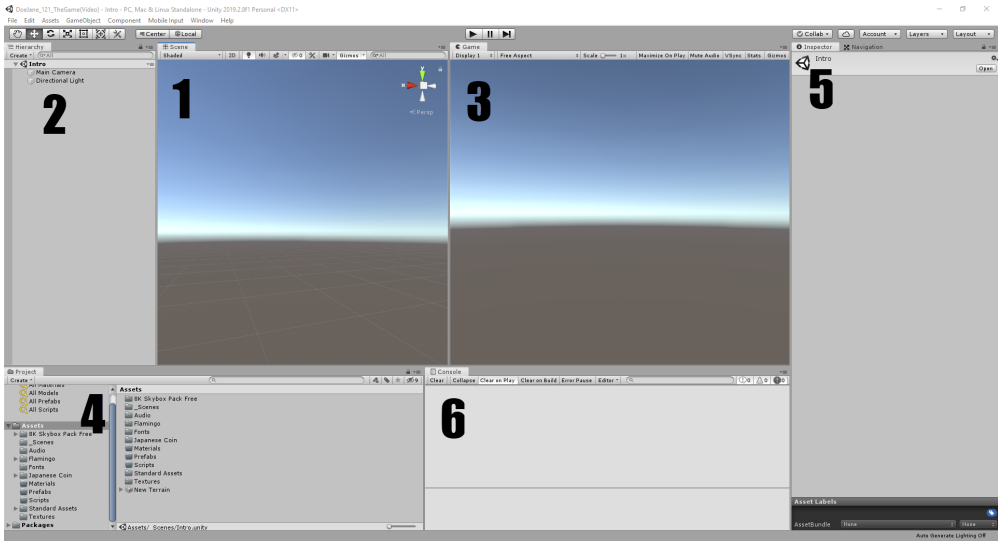
The Windows

The *Console Window* (#6) is the place where Unity “Prints” for scripts and provides messages when there are errors and warnings within the game. Trust me.





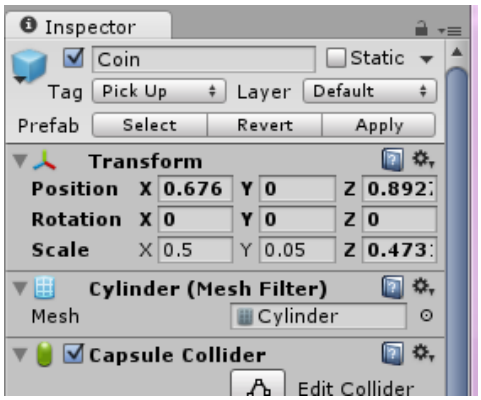
The Hierarchy Window (#2). This will default to a Main Camera and usually the Directional Lighting. The Hierarchy “houses” everything added to the scene.



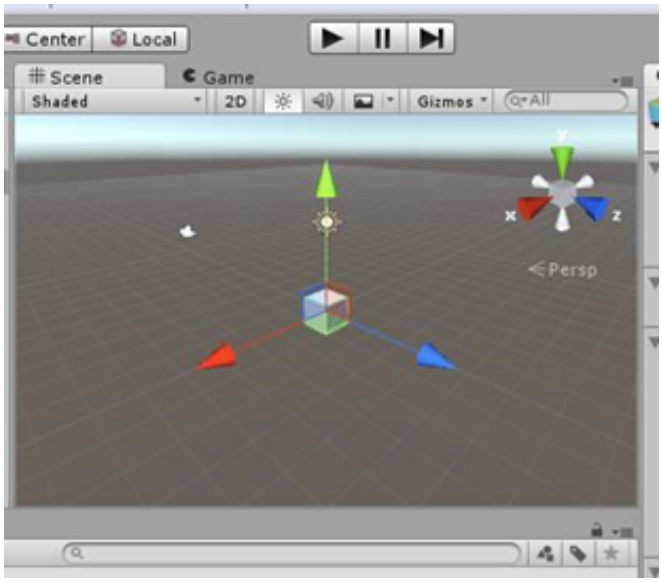
Let's add an object just to see what happens.

Navigate to the Menu and Select GameObject > 3D > Cube. Unity places a three-dimensional object on your screen. The CUBE now appears in the Hierarchy. You can rename this in the Hierarchy or within the Inspector.

With the Cube Selected, let's move on to this Inspector Window.

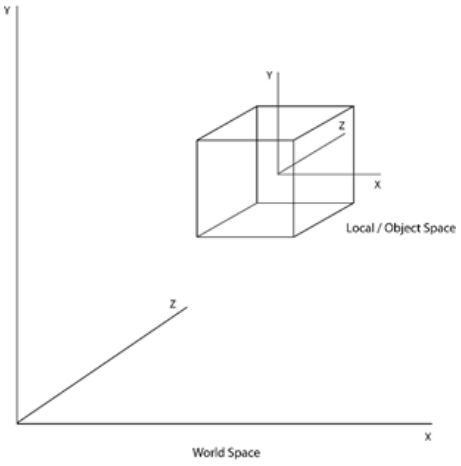


The Inspector Window (#5) shows all properties associated with the selected game object. Here you can change the position, scale, or rotation of the object as well as so many other "things". We are keeping it basic here so let's leave this alone…



Moving on to the Scene “View” (#1). This is the Scene. Everything you add here will be a part of the Unity Scene you are working with. Within the Scene Window, there are a few options, the draw mode of the object, the Render Mode (Shaded in Image Example), 2D or 3D Mode, Lights, Effects, and Gizmos. *The Gizmos can change the view to work on different axis of the object.*

Select the Cube within the Scene “View”. Hover the mouse in the Scene Window, and *Select the “f” key to zoom in*. Let’s take a look at the Inspector Window for the Axis information.



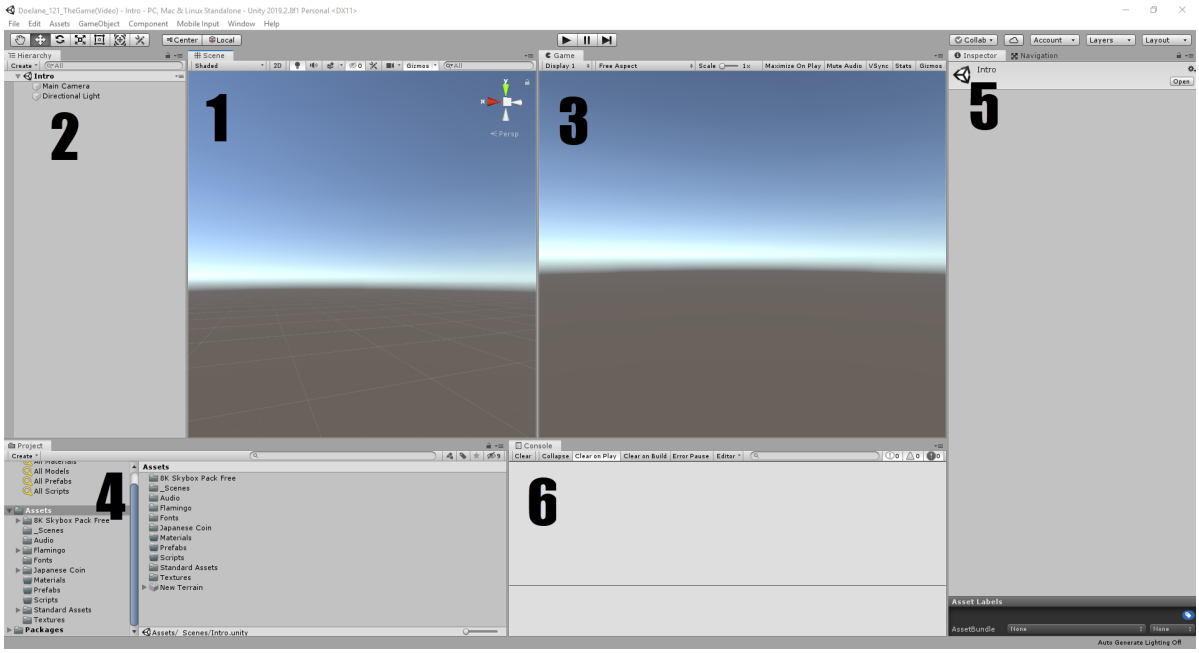
To simplify this “X, Y, and Z axis” in the 3D environment…

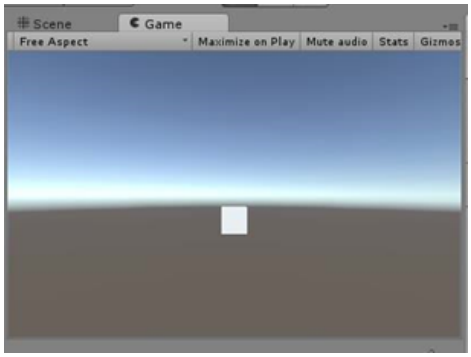
- X moves Right and Left.
- Y moves Up and Down.
- Z moves forward and backward (or near/far).

This process is far more in depth; however, this is all we need to know for now.

Let’s keep moving.

The Game “View” (#3). This is the view you have when the game is in “Run” mode.





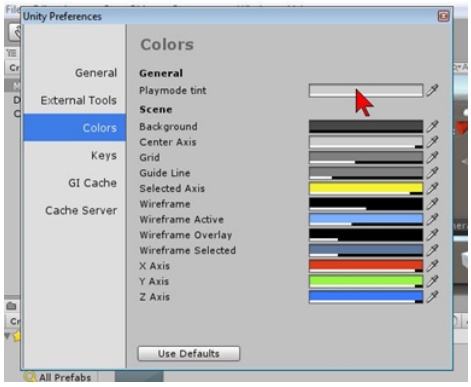
One BIG thing to mention here…OK …TWO big things to mention.

Thing One

When in Game Mode (or Run), ***any changes made to the project will not be saved***.

Because of this, I usually change my Unity application to display this horrible color around the other Windows when in Run Mode. This way I'll remember I am in Game Mode.

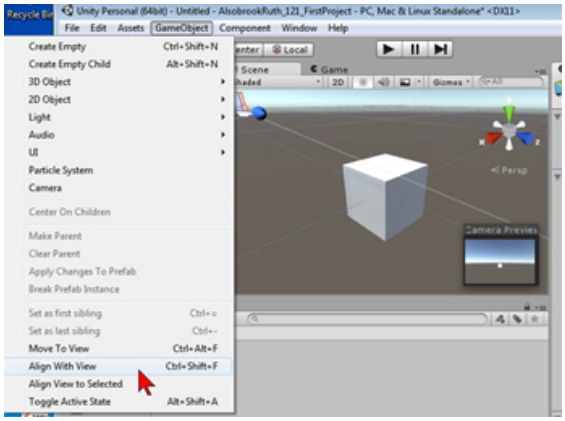
To change the Window Colors, just navigate to *Edit* > *Preferences* > *Colors*. Change the Playmode tint to something HORRIBLE or BRIGHT. You can also change any other colors here as related to titles.



Thing Two

When we select Run in Game “View”, there may be a different look than that of the scene. Go ahead, select Run and see what happens.

Do you see the Cube up close and personal? If you do, that’s great! If you don’t, that’s OK too.



What I am about to explain is a HUGE TIP which took me forever to learn.

- Exit Run Mode.
- Set the cube in the Scene Window in the center of the screen.
- Select the Main Camera.in the Hierarchy
- Select GameObject > Align with view.

Now run the scene.
Nice, huh?

Back to the rest of the “stuff” in the Unity Application; the tool bars.

The hand moves the scene around. Normal click is right and left, up and down, or Pan.
Left click rotates view. You can also Alt + Click here to rotate view.

Translate (w hotkey) – Moves objects

Rotate (e hotkey) – Rotates objects

Scale (r hotkey) – Scales objects other per axis or overall – Symmetrically.

The Rect tool is best used with 2D Sprites…Scales uniformly.

These are pretty much self-explanatory.

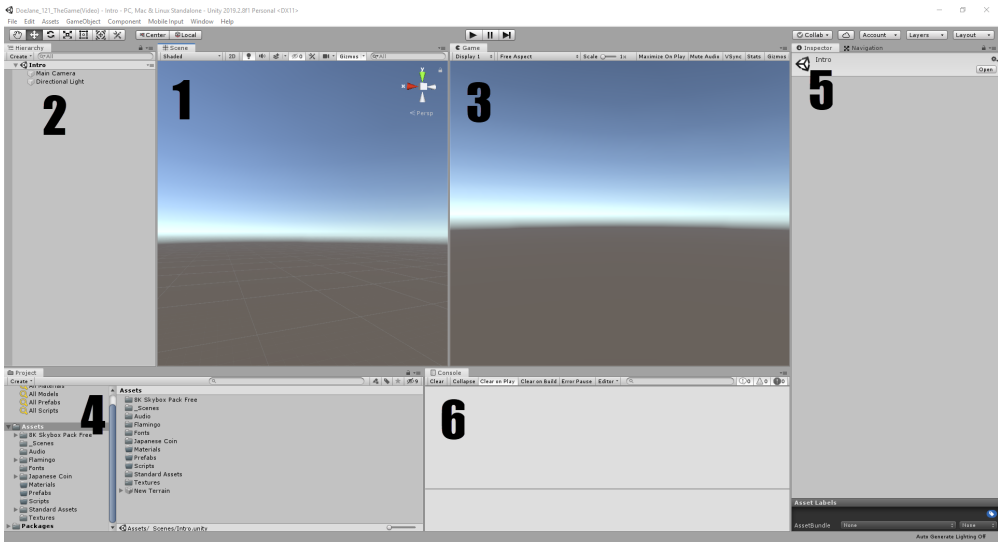
The Center and Pivot button represent where an object (usually a nested one) will move. Will it be from the Parent (main) or the Child (nested)?

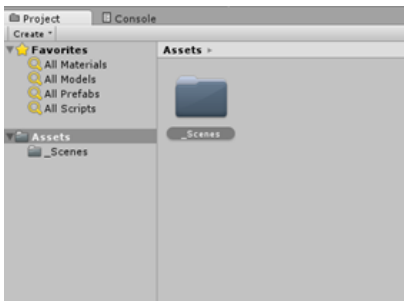
Local/ Global is whether an object reacts to a local space or a global space. Don't understand these things? Don't worry. You'll learn about them later.

Layers are used mostly with 2D work. Some 3D games also utilize this. If you are an Adobe Photoshop user, this is similar…

Layout…Well…You know about this one already.

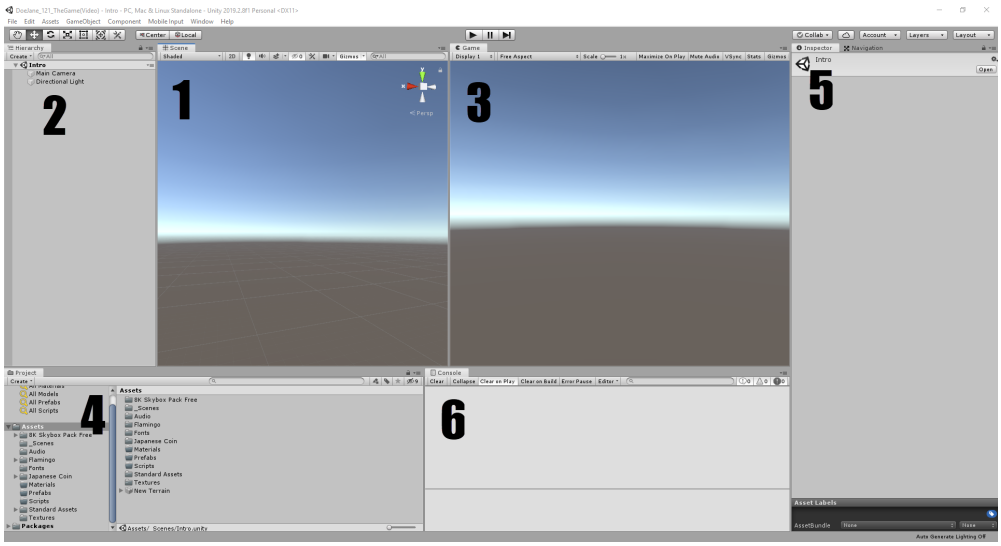
The Project Window (#4) includes everything added to the game project. This is where file management will come into play. Keeping this area organized is very important. You should already have one folder titled "Scenes" here.

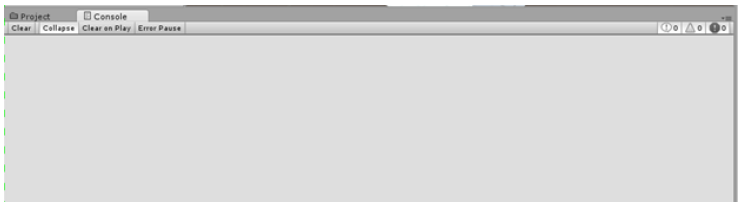




The Console Window (#6) is very important when you start writing scripts for the game. This is where you can test items by asking the script to “Print”. This is where you will see the print action.

This is also the area where warnings and errors will appear, if any exist. If the error is within a script, Unity will tell you exactly where to find the error, and provide possible solutions!





That’s about all there is to the interface for now.
https://youtu.be/D7duIqHFuiU
NOTE: This video shows Unity 2019; however, the information is still relevant.

Prepping for “The Game”

We will begin preparing our Game for Development. This Pick Up game will consist of an Island with the Third Person Player, The Pick-Ups, an Enemy, Audio, and whatever else you feel your island will need.

We’ll first begin by creating folders to house our “assets” for the game.

After this, we’ll create our own textures to use. This process makes the game YOURS. It brings into the Development originality; something no one else has in their game.

Next, we’ll find Audio which we can use for the Game Music, Pick Up sounds, footsteps, and environmental sounds.

Lastly, we’ll import our Unity Assets for use with our Game. This process will involve importing the Standard Assets, and the Enemy Package. Please be aware that if you choose to use a different Enemy than the Textbook, you may need to do some research to make YOUR Enemy work properly. Remember. The more assets you use, the larger the Project File, the Slower the Game Runs. Use Assets SPARINGLY.

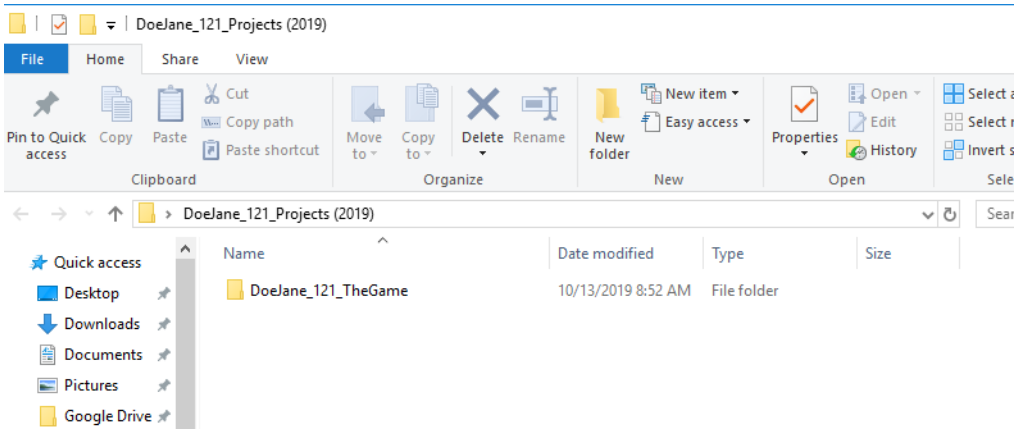
We will also need in your project, a document to keep track of all Assets used in the game. Name of Asset, Creator, and Where Found. This can also be kept on the Adobe Spark’s Page.

Let’s get started!

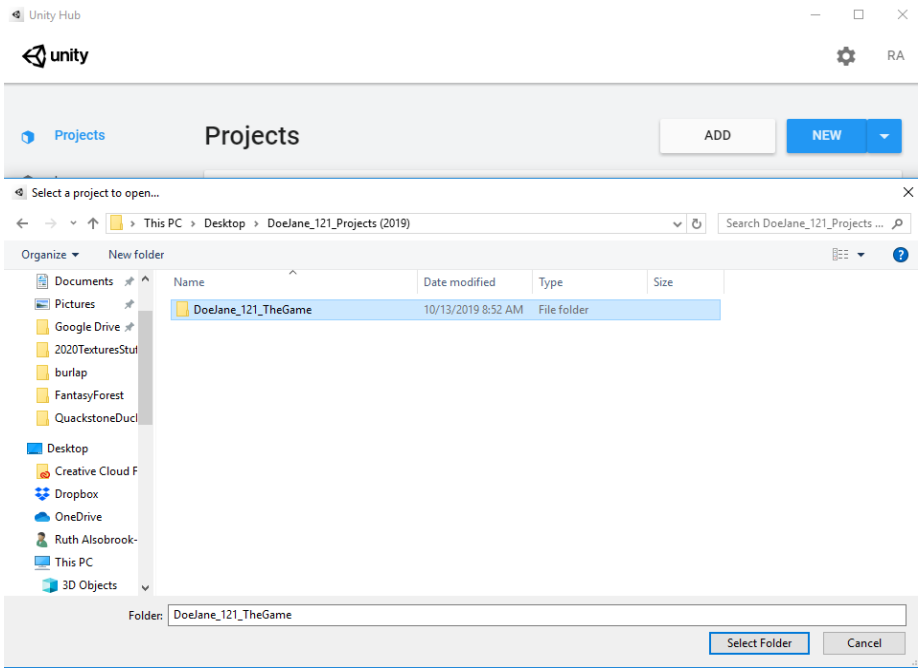
Folders

If you have not started The Game Project, let's do so now.

First, create the project folder in your main Projects Folder. Remember way back when we created this? Mine is located on my desktop. If you need help with creating a folder, think about the Right Click option ;-D



Open the Unity HUD, and select NEW. Select 3D. Name the Project (LastNameFisrtName_121_TheGame).
Select the three dots to locate your project folder. Select the Folder. Select Create.
Unity will open the new project.



If you have multiple versions of Unity installed, you may need to set this option within the HUD.

Double Click on the Project Name to open Unity.

Skip installing the New Version if you get the Download Message of a newer version.

Open the Scenes Folder. Rename Sample Scene to The Game by Right Clicking, and selecting Rename. Reload if asked.

Select the Assets Folder within the Projects Window. We'll add our folders here.
Right Click and Select New Folder.
Add Folders: Animation, Audio, Fonts, Prefabs, Materials, Models, Textures, and Scripts.
For myself, I also make a folder to hold any Assets I add to the fame (Added Assets). I also add an underscore to my Scenes Folder to the project, making this Folder show up as the first folder in line. These two latter folders are what I do, they are not mandatory.

We're ready now to start adding Assets!
https://youtu.be/lx-hn_cOjog

Models, Materials, and Textures

For the duration of this class, you will be building a simple Third Person, Pick-Up game. We will take everything you are learning and apply this in bits and pieces. It should prove very exciting. I look forward to seeing, and playing, your designs!

To start at the very beginning of YOUR GAME, we need to create a few textures. Don't be afraid of the word Create. This is something you can do. If creating is not your thing, guess what? There are TONS of FREE things on the Unity Asset Store.

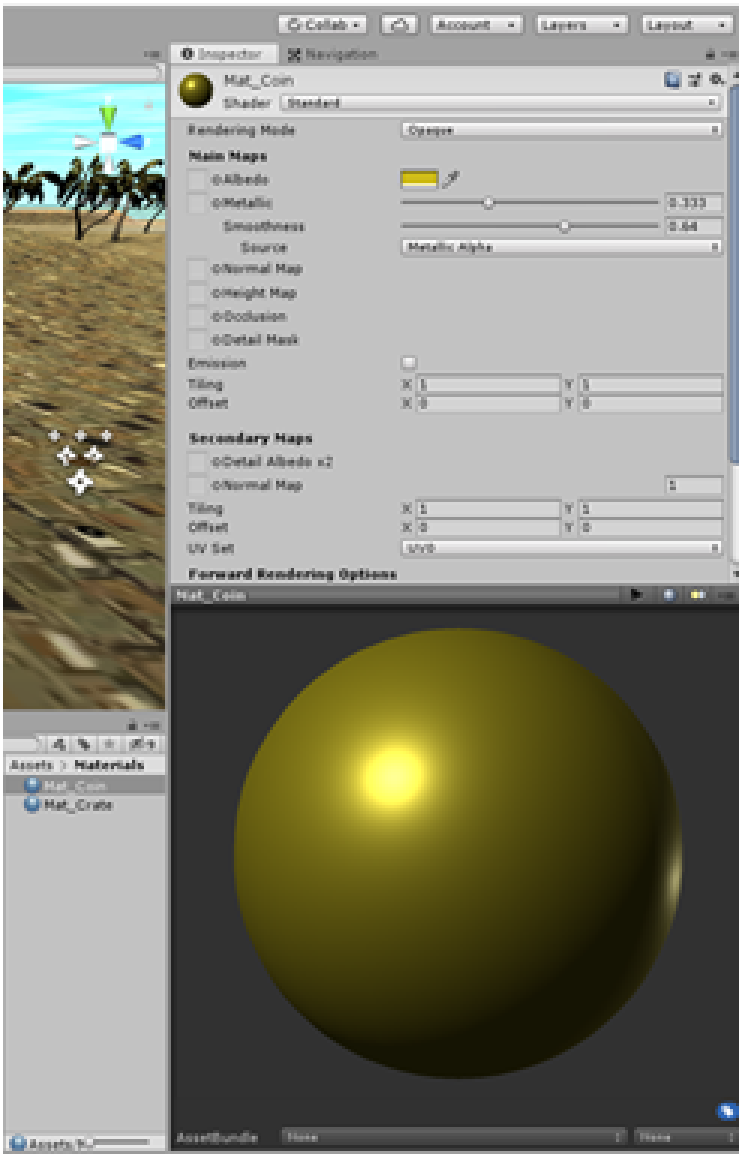
Be very wary of bringing lots of "stuff" into your game which is not needed when assessing the Asset Store. **The more you have in The Game, the higher the file size, the slower the play.** This is why planning a game is so important. Just know that every item you import from the asset store goes directly to the Unity application folder within your computer. It will be there for future projects.

Models

We'll dive deeper into models when we talk about the Asset Store, and when we start building our game. Models are .OBJ or .FBX files imported into your project. Be prepared to return to this great subject.

Materials

Materials are the creation of the textures used on the models. We'll cover this more when we get to the Building of The Game. Be prepared to return to this great subject.



Textures

I want you to create your own textures, *a minimum of three*. There is no need for costly applications such as Photoshop and Illustrator. If you have access to these, use them. If not, use a few free “things”.

Just keep the textures at a minimum size of 150 X 150, and a maximum of 1024 X 1024. For myself, I try to save as a PNG. PNG’s are Lossless data Compression. I just like it better this way. You may save your files as such, or with the JPG extension.

The following are tutorials on different ways to create textures.

GIMP

(https://www.gimp.org/)
Save yourself grief and review the Gimp Manual (https://docs.gimp.org/2.8/en/)

https://www.youtube.com/watch?v=Sayt-CLno9g

Wood Workshop

(http://spiralgraphics.biz/ww_overview.htm)
https://www.youtube.com/watch?v=KdiCIgOCs6g Wood Workshop
(http://spiralgraphics.biz/ww_overview.htm)

https://www.youtube.com/watch?v=KdiCIgOCs6g

Pattern Cooler

Free Online Seamless Texture Generator (http://www.patterncooler.com)
Please note: The gentleman from Patter Cooler has been offering FREE textures for YEARS. If you can afford a small donation, please provide one. Tell him Rah Rehula sent you.

https://www.youtube.com/watch?v=HWUgzSsB-EA

Photoshop

Since I am familiar with Photoshop, and can access this via Yavapai College, I will be using this to create a Crate Texture.

https://www.youtube.com/watch?v=e5bFwmiteVg
NOTE: Shift Key is no longer needed to create a perfect square. Just click and drag ;-D

Find more texture generators, see here (http://1stwebdesigner.com/pattern-generator-online/)

Create THREE textures for The Game, *using whichever method you prefer.* These textures can be Wood, Grass, Flowers, Sand, Rock, and Geometric Shapes. Create whatever you think would be fun to add; or try. Once these textures are created, be sure to add them to your Textures Folder within the Project.

What I usually do is create toward the environment I am building. Since we are creating an “Island”, my thoughts led to sand, rock, shells, and other beach things. If your island is a Fantasy Island, create toward this design. If you are thinking of a frightening, Halloween Theme, design towards this.

I’ll also create grass textures; being sure to use a transparent Alpha. If this is something you are not aware of, don’t worry. Maybe we’ll do a little video on this later.

Add the Textures to the Project once they are completed.
Double Click the Textures Folder. Click one texture, and Shift Click the last Texture to select all.
Drag and Drop into the Texture folder.
SAVE the Project.

https://youtu.be/-PVr2yl7kxA

Audio

We now have great textures to work with when we start developing our game. It may seem like we have done nothing; however, we have completed a great deal of work!

Our next find for assets is **sound.** For The Game, we'll need a background sound, an environmental sound, and a pick-up sound. This is why knowing what style of game you are creating is important. If this is a "ghostly" adventure, you'll want some spooky sounds. If you are going for a traditional island, you'll want some waves, and seagulls for the environment. Sound good? Pardon the pun.

Later. You may wish to add an explosion sound to the pick-up, and/or footsteps to your player, or the enemy.

You can even create your own sounds with FREE applications. The sky's the limit.

There are many places to find free sound. Heck. Even YouTube has some now. What is important, is to know the Creative Commons Attribution (https://creativecommons.org/licenses/) to the audio you are using. Be sure to keep track of who, what, and where, to add to your Assets List for Source Citing.

Find yourself a Background Soundtrack, an Environmental Sound, Pick Up, and a Footstep Sound. In this textbook, I'll show you how to create a Coin. I'll add some links to coin sound for you. If you are thinking of a pick-up for your game, search for this sound.

Once you have your sounds, drag and drop them into the Audio Folder, just as you did with the textures. Save the Project.

Background Music
This is a harder audio to find for FREE. I mean, there are great sites out there. For myself, since I create a new game every time I teach this class, I decided to create an account with Storyblocks (https://www.audioblocks.com/). Here, I can purchase music and sound bytes. A "one stop shop", so to speak.

I am not asking you to create an account or pay for anything.
Find something for FREE. We will be going over the Unity Assets Store in a few. This is a great place for free audio as well. If you'd like to read up on this, go to the Unity Assets Store Chapter.

MUST CREDIT SITE ;-D
https://patrickdearteaga.com/royalty-free-music/
https://www.purple-planet.com/
https://soundimage.org/
https://incompetech.com/
http://openmusicarchive.org/

https://musopen.org/
https://freesound.org/
http://www.flashkit.com/soundfx/
http://ccmixter.org/

Coin Sounds
https://www.freesound.org/browse/tags/coin/
http://www.soundsnap.com/tags/coin
http://www.orangefreesounds.com/mario-coin-sound/
http://www.soundjay.com/coin-sounds-1.html
http://soundbible.com/2081-Coin-Drop.html

Environmental Sounds
http://soundbible.com/tags-beach.html
http://soundbible.com/tags-island.html
http://www.listeningearth.com/LE/general.php?pageID=8
http://www.partnersinrhyme.com/soundfx/Ambience.shtml

https://youtu.be/99ceIV8nwU0

Fonts

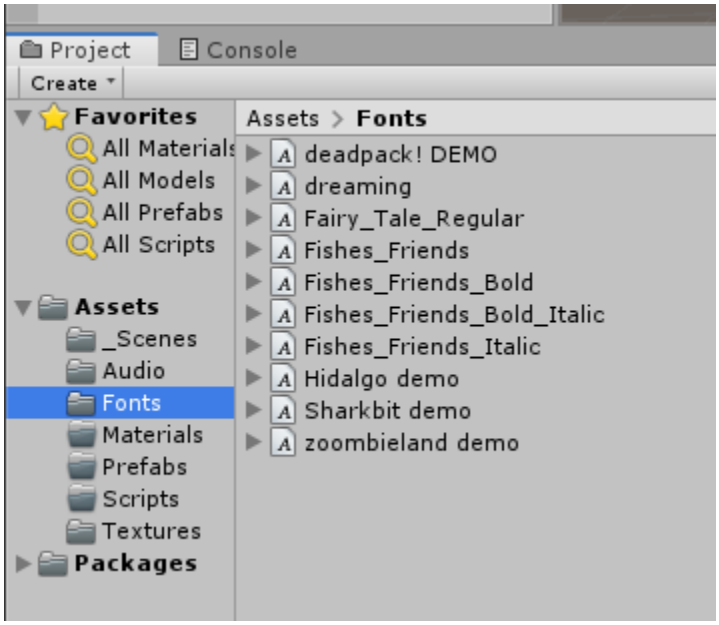
The Game will consist of an Introduction Scene with a title, directions, a play option, and a quit option. When play is selected, The Game Scene will open. If the player wins, a Win Scene appears with a message of winning, a replay option, and a quit. If the Player loses, a Lose Scene will appear with a message of loss, a replay option, and a quit option.

These scenes will use a UI (**User Interface**). We'll want some special fonts to go with the theme of your design. You should have some idea of what genre you are going with by now. The cool thing is you can download special fonts to use in your game.

First, you'll need to understand the process of finding your fonts. If you are on a Windows Machine, this site will assist -
https://support.microsoft.com/en-us/help/314960/how-to-install-or-remove-a-font-in-windows
If you are on a Mac, this site will assist - https://support.apple.com/en-us/HT201749

Now that you have a better understanding of the Fonts Folder on your machine, check out these sites to find fonts. I'd go with THREE for the final design. I'll probably add about seven to test in the project.
https://www.dafont.com/
https://www.1001fonts.com/

After downloading your fonts, click and drag these to your projects Font Folder.



https://youtu.be/SmI2tEFG5DQ

The Asset Store

The Unity Asset Store is amazing! You can find so many options within here for your game. With that said, a few things to understand is the more you have in your project, the larger the file, and the slower the game play. Very important to understand this. Another “thing” is to use FREE STUFF when you first start out. Please, save your hard earned money. I am hoping to show you how to create a personalized project for FREE. So far, you’ve used free resources to create textures, find sounds, and find fonts...all for FREE.

Because our game is a third person pick up game, we need to import an older asset with controllers. We’ll also need a few particle effects, and a camera script. We’ll also need to find an Enemy for our game genre, and possibly a pick up item. For this latter object, I’ll provide a simple coin design later, using Unity Primitives (prims - Game Objects).

I’ve done a lot of the legwork for you in importing only what we need with the Older Standard Assets package from within the Asset Store. This used to be the go to for this project. However, with all the newer versions since 2017, some of the features fail. So, I imported only what was needed, and corrected a few scripts, then Exported a package for you to use in your project. Find it here (Standard Assets Package).
NOTE: I dragged this to my Added Assets Folder for File Management

Once you have saved this package to your computer (DO NOT EXTRACT), select Assets > Import Package > Custom Package. Once the package has “decompressed”, select Import. Unity will import this package into the project.

Because this package is from an older version of Unity, there will be warnings and errors which appear in the Console Window, just clear this and make sure NO ERRORS remain. If they do, and do not clear when clear is selected, you’ll need to contact me ASAP so we can fix these. Select the Run/Play feature to be sure no errors pop up. If you do not see a Compiler Error Message, you are good to go!

SAVE the project.

https://youtu.be/YE3Jt-SIS7E

Now, let’s take a look at the Asset Store. If this is not open within Unity select Window > Asset Store. You will need to be logged in in order to “purchase” any assets. Once logged in, think about your game genre. I’ve built this pick up game so many times. Most of the time I go with humorous and vibrant colors, I think I’ll try something different this time, and go with a “dream” style theme. Now this leaves my door wide open on what I can use for an enemy and pick up items.

Now, like it was stated before, **DO NOT PAY** for anything. Try to find **FREE** things. We'll first search for an Enemy. For your first game, you wish to stick with Humanoid Assets. The textbook enemy example will be an Iguana (https://assetstore.unity.com/packages/3d/characters/animals/reptiles/iguana-57458) . You may use something different. To narrow the asset field, select the filters option, Select 3D > Characters, scroll down and change the pricing feature to Zero. Because I am using Unity 2020 in the example video, I am going to select this feature as well. Next "View Options". You can also select "Search" and type a specific thing to look for…

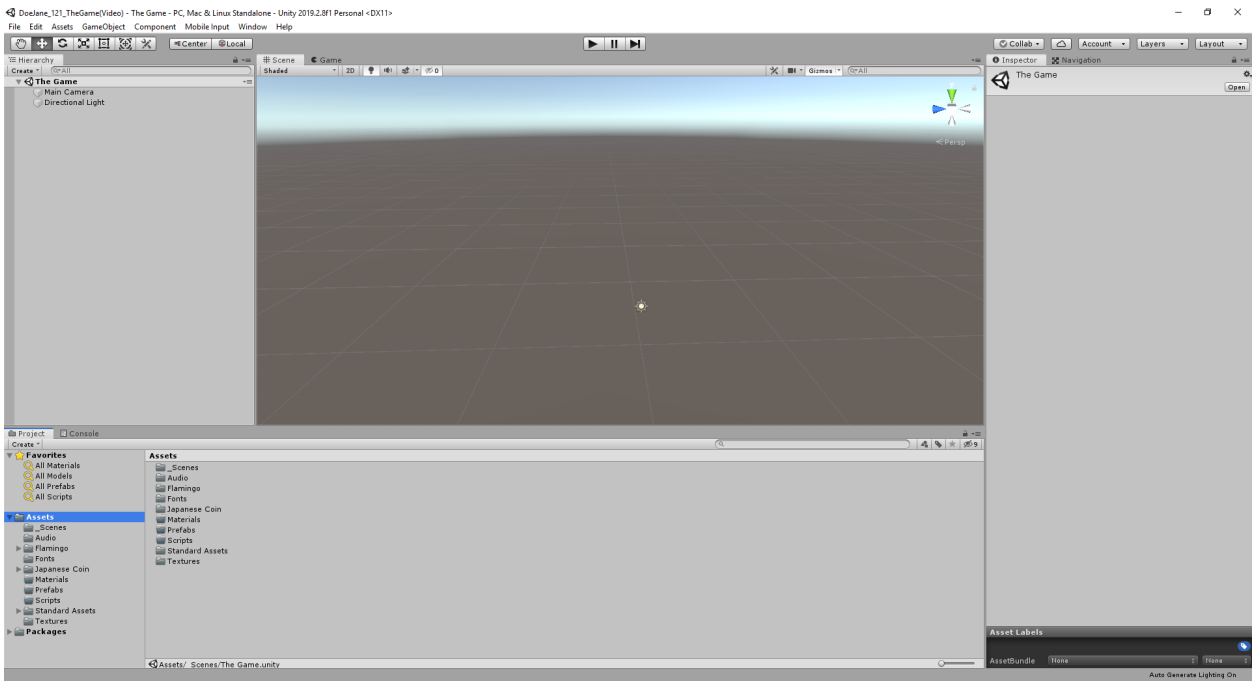
Once you have found your "Enemy", Download and Import. Do the same for a Pick Up object, unless you want to wait and learn how to create your own coin!

Once everything is imported, select Run/Play to be sure no Compiler Errors appear. If none do, you are good to go! Save your Project.

https://youtu.be/YE3Jt-SIS7E

If you wish to close the Assets Window, just right click on the tab and select Close Tab.

We now have our preparation completed. We can begin building the game. This is where the project comes to life! You'll see where all your hard work in this section will pay off my friends!



Our Project so far...

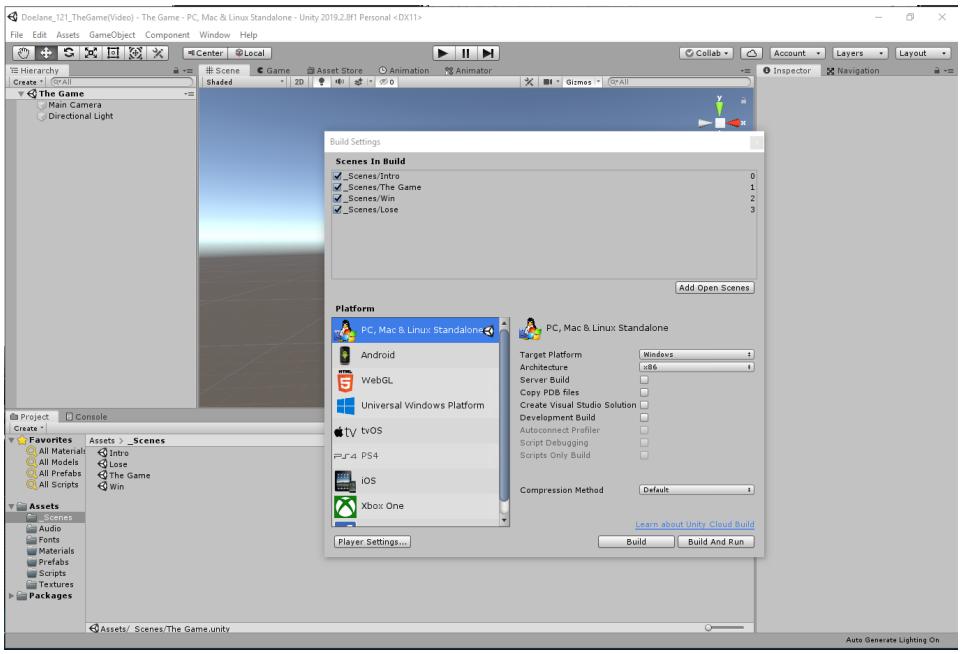
Scenes

Since we know we'll have four scenes in our game (Intro, Game, Win, Lose), let's go ahead and create these within our scenes folder. A very simple process of Right Clicking. I love the right click process. You'll soon find that to be the case too!

Open the Scenes Folder within the Projects Window. There should be a Sample Scene. If you have not done so, rename this scene to The Game. Next, Right Click inside the folder and select Create > Scene. Do this process until you have Four Scenes. Name the Scenes Accordingly (Wine, Lose, Intro).

One last step to set up our game's build settings. Navigate to File > Build Settings…
Click and drag the Intro Scene to the Scenes In Build. Next, add The Game Scene. Finally, add the Win and Lose Scene.

Most important here is the first Scene; Intro. This is how The Game will begin. Next in line is The Game Scene. Once the Intro Scene opens, and Play is selected, The Game Scene will open. The Win and Lose order does not matter in this project.



When you open a Scene, details show in the title of Unity. Be sure to select The Game Scene before saving the project.

https://youtu.be/wuv4uPHxxOw

Start Building The Game

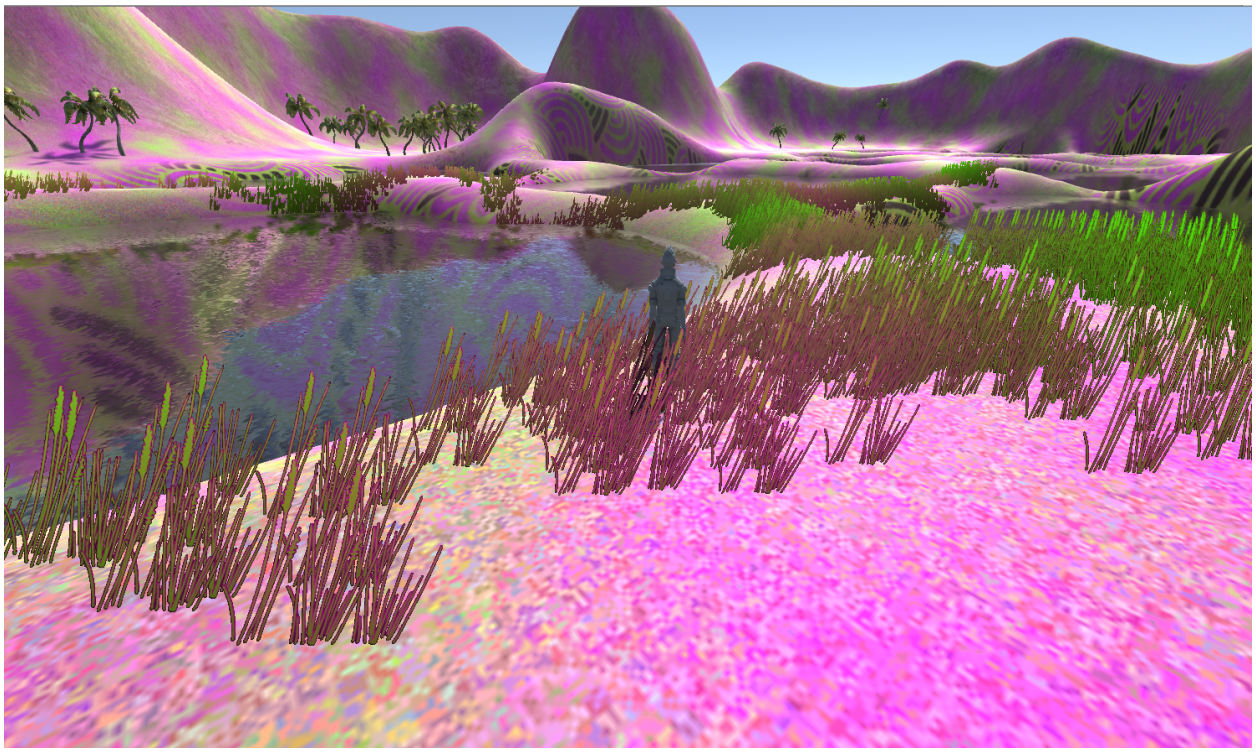
We have now prepared our project to start building our Pick Up game. Everything we need, minus our scripts and builds are within the project. The first thing we will do is build an island, and use our beautiful texture assets we imported. Secondly, we'll add our third person character to go around the island and Pick Up the objects. Afterward, we'll add an enemy, some Level Manager "stuff" to advance the levels, as well as User Interface (UI) text for interactivity (play, quit).

Let's get started!

Environment

As stated, we are building a Third Person Pick-Up game. We need an environment for this character to move about. We are going to go with an Island feel. Think about what genre you are leaning toward when building your island. Will your world be realistic, or fantasy? Will the game be frightening? Will it be funny?

Another huge thing to think about is where your character will run about the island. Sometimes, folks start building their island, and they forget to utilize areas for walkability. Be sure to add areas that are specific for trails or paths.



Terrain

Before you begin, make sure you have The Game Scene opened in the project.

There is a little bug in some versions of Unity which causes the Application to Freeze when you start working with terrains. It has to do with an “Auto Generate” of the lighting source. To turn this off, select Window > Rendering > Lighting. Within the “Scenes” Tabe of the Lighting, UNCHECK the Auto Generate.

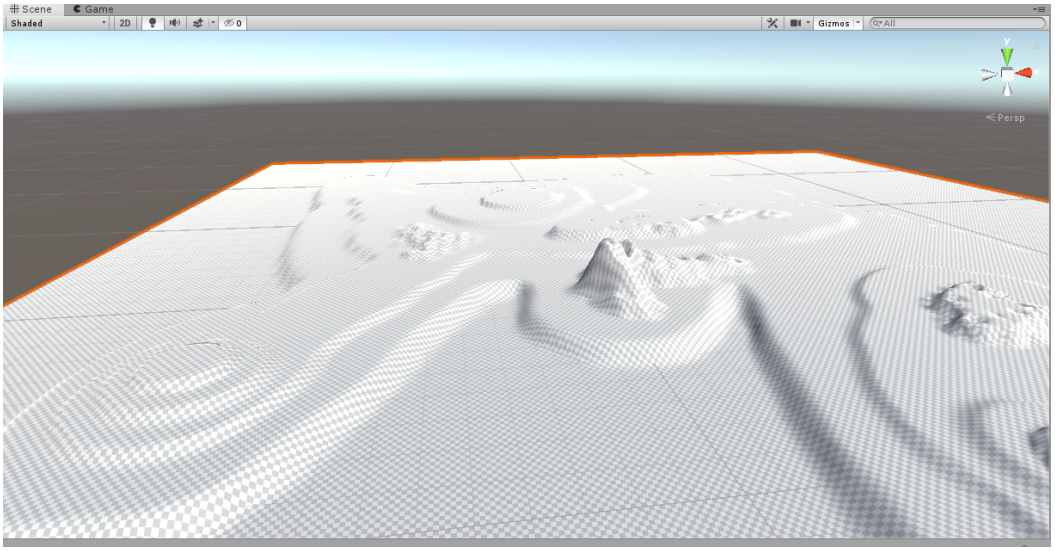
From the menu, select *GameObject* > *3D>Terrain*.
This slaps a Terrain right in the middle of the Unity Scene. How simple is that?! All you need to do now is to resize and start molding this Island to your liking.

Before we begin “molding” the Terrain you need to think about the outcome of the game.
This game will be in Third Person and will consist of the player “picking up” objects throughout the playing field. Once all the objects are collected, you win. Pretty simple.

So now, think about how you want your character to move about the terrain. Do you want them to only move around the beach knolls of the island or move into hidden areas within a mountain? What about on top of Mesas? Maybe the character walks upon a path and selects a bouquet of flowers.

There are so many possibilities to what you can create; at times, it can be overwhelming. Since this is your first game, try to keep the end result simple. For this class, you will barely get your feet wet in the scripting of game objects. In fact, we will probably create about four scripts for our game!

Remember to keep the terrain and game idea as simple as possible. This will allow you to come back and build upon your basic design once you gain more experience.



It is also fun just to come back and review simpler times.

Now let’s get started with the Terrain.

First, the default size of the Terrain is HUGE! We are going to

create a small island from this HUGE Terrain, and use the rest of the Terrain as the Ocean Surface. Still, we do not need such a large terrain. Let's reduce this a little.

Unity's terrain is defaulted at the size of 1000 meters squared. This far more than we need. In fact, half that size is still big. Let's start with 100 square meters. You may go larger if you like. I would not go above 500 square meters

To change the Terrain size, select the terrain within the Inspector Window. Next, select the COG. Scroll down and change the Terrain Width and Terrain Length to 100 each (or your chosen size...Please do not go larger than 500 square meters).

You can leave the Terrain Height at 600 or change to 300 as the video shows.

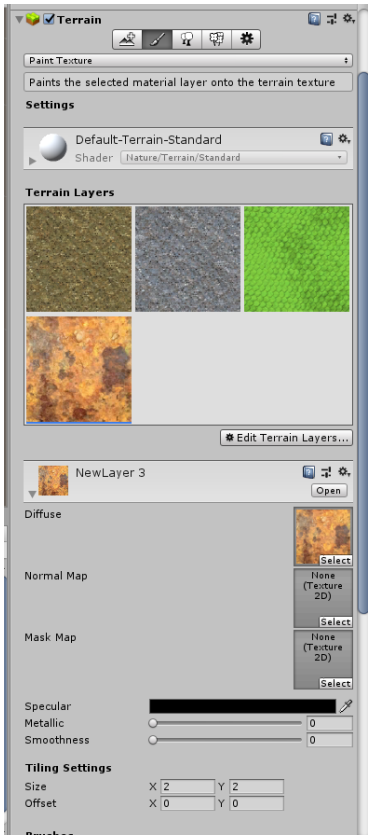
Now, let's work on the shape of the island.

Select the Terrain within the Hierarchy. In the Inspector Window, select the Paint Brush to create the Base Height on the Terrain. This first step will create the shape of the island. Choose the Set Height from the small dropdown menu. You should find this right below the paintbrush. Set your Brush size to about 39, and the Height to 5. Paint the Basic Shape. Once this is completed, go back and add more Height for Hills and Mountains. Each time, increase the height, and change the brush size, and the shape of the brush if you like. Use the dropdown menu to select Smooth if you'd like smooth edges on your terrain. This is best around edges to prevent sharp points within the terrain.

https://youtu.be/zzQ9jcdwQcs

Painting the Terrain

Now, we need to paint the terrain. This is where we will add our wonderful textures, which we created earlier and imported into the project.



With the Terrain selected, Select the Paint Brush, and choose Paint Texture in the dropdown menu. We'll need to add some textures to use here. Within the Inspector Window, select Edit Terrain Layers > Create Layer. Double click the texture for your base paint. Many folks use the Sand option here. Once you have double clicked, the whole island will have that base layer added.

If this texture does not look right to you, changes can be made to the texture repeats.

- Select the texture square within the Terrain Inspector Window.
- Scroll down to Tile Setting. Within the Size, change the X and Z. These numbers should stay in the same counts as Unity recognizes (32, 64, 128, etc.).

Let's add more terrain textures and paint. Follow the same steps above to paint the rest of your terrain textures (sand, rock, grass, etc.).

Be sure to use the keyboard shortcuts to move around your terrain to paint…

Remember to change the Opacity of the paint as you move upward. This helps to blend the changes and create a more "natural" Terrain.

Save your work often.

You can find more information on the Terrain here
https://docs.unity3d.com/Manual/terrain-OtherSettings.html

https://youtu.be/Kacgpa8wuqw
NOTE: I click ONCE for the base layer, thus making this a "paint" option. Would have been quicker to double click!

The Terrain looks pretty good so far. To make it come to life, we need a few features.
You will add trees, grass, sky, light, and some fog to make the environment come alive. Lastly, we'll add a third person to the terrain so we can move around. Go ahead and open this Project if you have not done so, and we can get right to it!

Trees

In the Terrain Paint Tool area (Inspector Window), select the Tree Brush. Remember selecting the Textures to paint the terrain? We'll do the same thing here for the trees. Click *Edit Tree* > *Add Tree. Select the small circle in the dialogue box to add a tree.* In the search option, you can type a specific tree, such as Palm, if you know what you are looking for.

Select the Palm tree (or your choice of tree) in the list, double click, and then click "Add".

Before we paint, let's look at the setting for the trees. The **Brush Size** will determine how large the trees will paint when you click on the terrain. **Density** represents how many trees will appear with one stroke. Think Forrest (high number) versus park (low number). Do you want the width and height to be symmetrical? If so, keep checked. If not, uncheck and set the width to your liking.

What about variation of color? Do trees usually keep the same color amongst them? Do trees vary? How much variation do you want? What about rotation? Do all trees face the same way? Moving down, think about a Collider. What is this "Collider"? This is whether an object will hit the tree and stop, or move right through it. Let's leave this for another time. If you feel daring, go ahead and try this tool.

Once you have set up your desired numbers, start painting the terrain. If you wish to remove a few trees, *select the Shift key and paint*.

https://youtu.be/W_0Vzm6KrVA

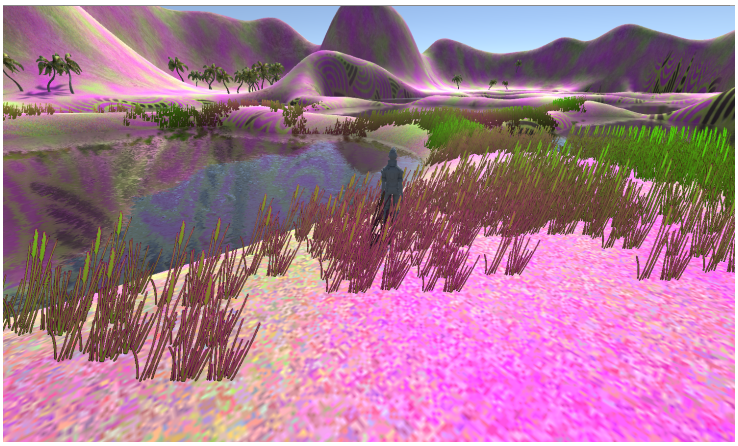
Grass

Grass works the same as the trees. Just select the next “brush”. Scroll down and Select Edit Details > Add Grass Texture. Once again, select the circle (you’ll get used to the circle, or should I say “Radio Button”), and select the grass you prefer. Grass Frond is a default for the Unity Application. I am going to use one of my own textures. Select the colors of your grass. Maybe you are going for a fantasy theme…Purple and blue…

Select the specific brush size, opacity, and target strength and start paining the grass. You’ll need to zoom in real close to see the grass being painted in the Scene.

Too much grass? Hold down that Shift key and paint to remove.

What’s really cool about the grass is when you see it in Run mode, the grass will move with the wind elements. It is just amazing to see. Way back when, it took far more than just a swift paintbrush to get grass on a terrain; and to make it move with the wind?! Lot’s of Scripting involved. Now, you just paint, and Viola!



To see your beautiful grass swat in the wind, move the scene view to somewhere which shows the grass. Once positioned, select the Main Camera. Go to GameObject > Align with view. Select Run, and see your grass move in the wind.

NOTE: *Video Shows the alignment of camera at end...*
https://youtu.be/Urra_2trJ7U

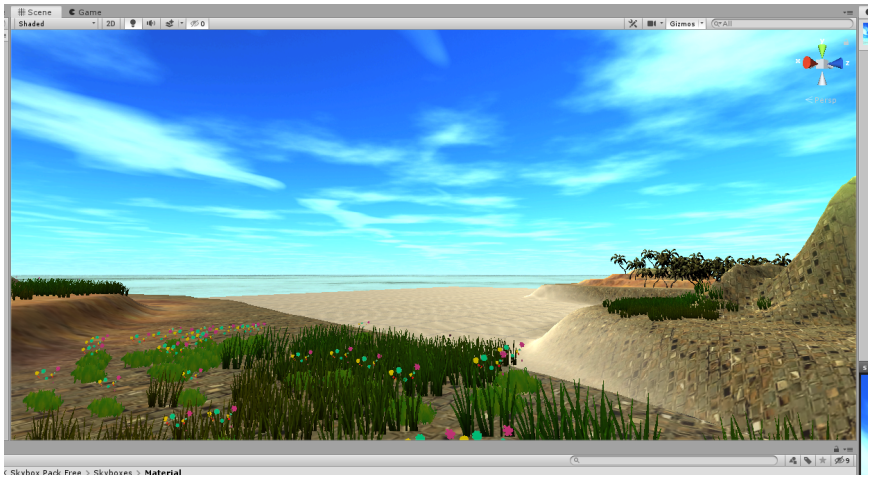
Skybox

Now that there are trees and grass on the terrain, we need a beautiful sky to look at! Right now, the sky is set at a default setting. You can leave this as is, or try something new. The choice is yours. If you do not like what you do, you can always go back to the default setting. As stated, you can leave this as is, and move to the next section.

You can choose to just use a color for your sky. This is a simple process of changing the Camera view within the Inspector Window. Select the Main Camera within the Hierarchy. In the Inspector Window, change the “Clear Flags” to Solid Color and choose your color. Sometimes, this choice can prove quite interesting!

You can also create skyboxes.To “vamp” up the Skybox, we will need Skybox textures; a Skybox Asset. Guess where we go? You already know how to import Assets.
ONLY DO IF YOU WISH TO TRY THIS ONE…Not Mandatory.

Open the Asset Store (Window > Asset Store). Type “Free Skybox” in the search. Select a FREE Skybox package. For the video demo, I already installed the 8K Skybox Pack Free by BG Studio.



Materials for the Skybox came created within the 8K Skybox Pack Free by BG Studio package. **IF THEY HAD NOT,** I would need to create the Skybox Material with these steps…Steps we will cover again.

Open your Materials Folder. Right click and create a new Material (Material_Skybox). In the Inspector Window, select the dropdown for Shader. Choose Skybox > 6 Sided. Choose the corresponding images for the textures by clicking and dragging the images to the correct “squares” for textures.

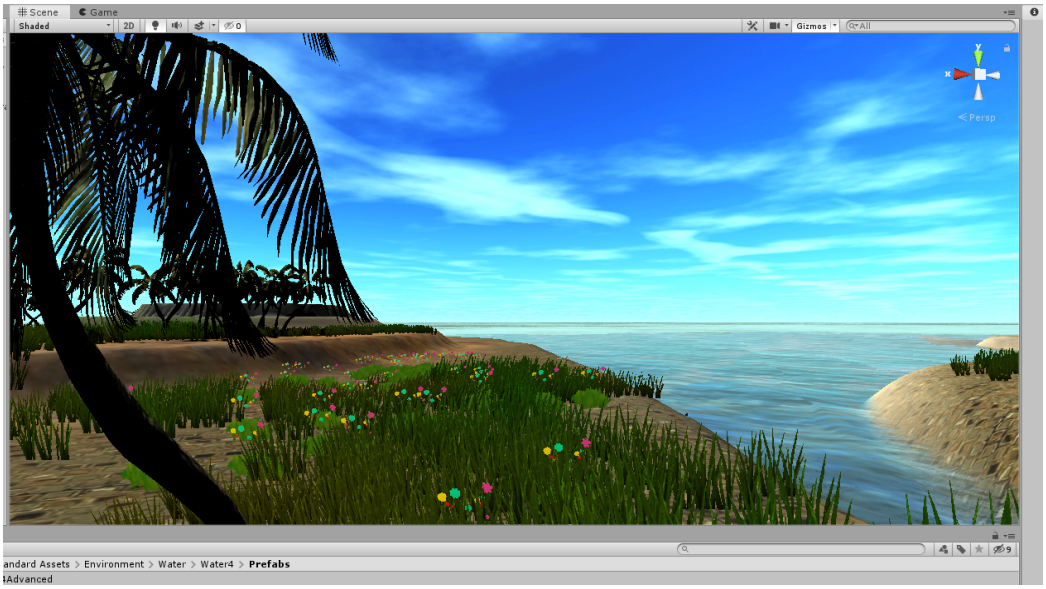
You can even change the Exposure and Color of skyboxes! See what the Skybox will look like in the preview window.

To add the new, gorgeous, Skybox to your Project, go to Windows > Rendering > Lighting. Select Environment Tab.
Find the small circle and choose your skybox (Material_Skybox).

Look at that sky now!
https://youtu.be/1uogoYhmYS8

Water

When we imported the Environment Asset, Water came along with this. Wait until you see how simple it is to add water! Find the Environment folder. Open it.

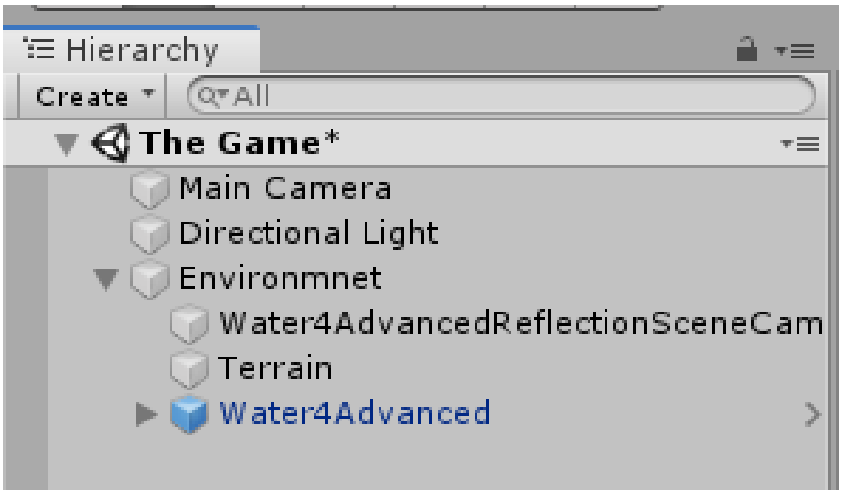


Open the Water Folder. There are two types of water. Open one of these and then open the Prefab. Click and drag the Prefab Water to the Scene. Resize the X and Z Axis ONLY. In the video, I use the scale tool. You can use the scale Transform Numbers within the Inspector Window.

Zoom in on the water, be sure to have a little grass showing, and change the view of the Main Camera to “see this” *(Select Main Camera in Hierarchy > GameObject > Align with View)*. Now Run the Game and see your water and grass movements!

https://youtu.be/jJq40CHJagw

Environment Wrap Up



Lastly, we need to place all our Environmental objects into one EMPTY Object. This step is very simple. Within the Hierarchy Window, select Create > Create Empty. NAME this Empty Object “Environment”. Click and drag the Terrain and Water into the newly created Empty Object. These are now Children of the Environment Empty Object.

Many ask why we do this. It’s all about organization. We now have all the environment objects in one area. When we do any work here, which we will, it’s now a one click process.

To put it plainly, this process is done for organization and simplicity.

Character Controller

We now have a beautiful world and need to be able to roam about it and enjoy its beauty. This is where the Character Controller comes in. First, let's take a look at the difference between First Person and Third Person.

The Third Person allows us to see a character while walking about. This visual can be from behind, beside, in front, or above the character.

The First Person is just that. You will roam about with your eyes only. The game is played as if you are the main source of movement and steps.

Within the Standard Assets Folder, Open the Characters. Open the Third Person Folder. Read the Guidelines! I wish for you to become acquainted with how Unity operates the Third Person. We will use the plain old Ethan here. If you have not added Ethan to the Island earlier, do so now. Simply drag him to a spot. **Use the ThirdPerson Prefab** for now; ***not the AI***.

For now. Nothing will happen when you start the game. What I mean, is that the camera will not follow your character. Let's fix that now.

Select the Main Camera in the Hierarchy Window.
Go to Component (Inspector Window) > Scripts > UnityStandardAssets.Utility > Smooth Follow.

This adds a Smooth Follow script to your Main Camera and tells it to follow a particular object. We must tell Unity what object to follow!

Within the Script Component (Inspector Window), select the Radio Option next to **Target** and choose **EthanBody** in the Dialogue Box.

Change Distance to 5, Height to 1, Rotation Damping to 3, and Height Damping to 3. Now these numbers are personal choices. You can play around with the numbers to find what you like for your game.

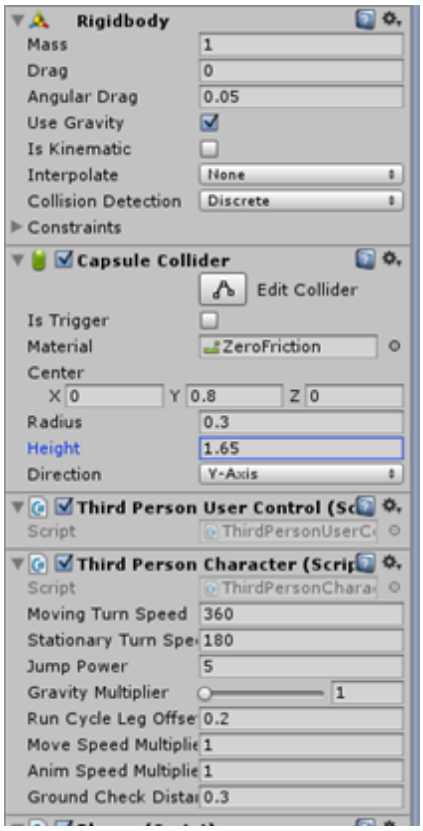
Select Run.
Use the Arrow Keys to move the Model throughout the Terrain.
How cool is that?!
Exit Run Mode.

SAVE your Scene and Project.
https://youtu.be/0EOOsIuUR4Q

The Controller

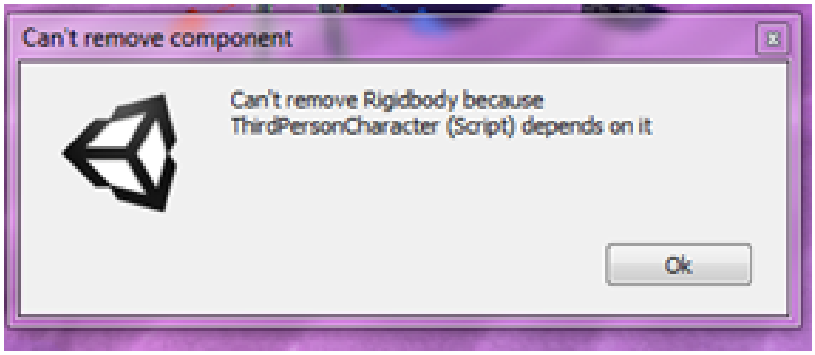
A Character Controller is what controls the Third Person or First Person Controllers in a game. It is utilized when the object does not make use of Rigidbody physics. In fact, the Unity Manual states just this! Check it out HERE (http://docs.unity3d.com/540/Documentation/Manual/class-CharacterController.html).

We are going to dive into some of the features Unity offers us with the Character Controller components. In your game, select the Third Person Controller to open the Inspector Window. We'll take a look at these



The first thing to note is the Rigidbody is set on this Character Controller. WHAT? This goes against what Unity just told us! Yep. Things like this happen all the time.

The reason this component is there is due to the ThirdPersonCharacter Script. This script calls upon the component to operate. Go ahead and try to remove the Rigidbody Component by selecting the Cog > Remove Component. You get the following message:



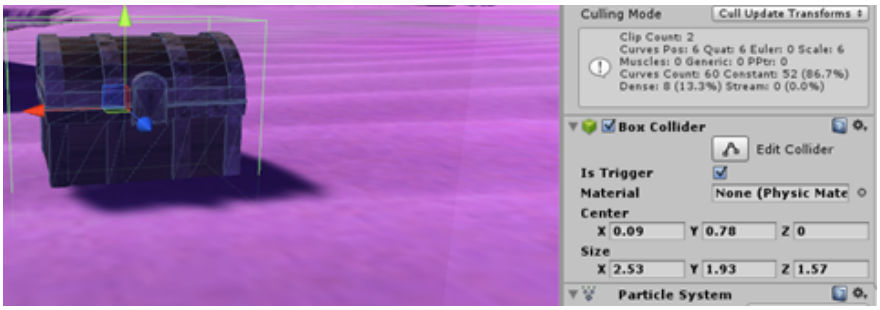
If you open this Third Person Character script in the IDE (integrated development environment), you will see on line 5 “Required Component”.

```
using UnityEngine;

namespace UnityStandardAssets.Characters.ThirdPerson
{
    [RequireComponent(typeof(Rigidbody))]
    [RequireComponent(typeof(CapsuleCollider))]
    [RequireComponent(typeof(Animator))]
    public class ThirdPersonCharacter : MonoBehaviour
```

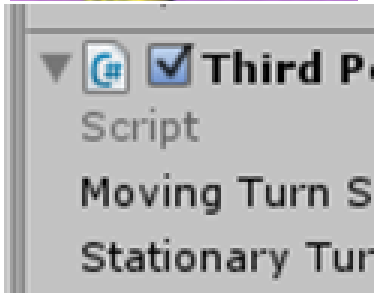
We see from this that the basic components of Unity can be adjusted by using scripts. Imagine that?!

Let’s get back to the Inspector and look at the next Component; the Capsule Collider. This feature is the capsule which wraps around the Third Person. It is what is called upon when the Character “collides” into something. For us, at this point, it is the Pick-Up items.

The X, Y, and Z Center is what associates the collider with the object. If you were to make this collider smaller, it would make the “collide” with the Pick-Up more difficult. The Pick-Up object will also have a collider.



If needed, select the Third Person Controller. Within the Inspector Window, we will look at the Components associated with the Third Person Character script. If this is collapsed, select the arrow on the left to open.



If you open the script in the IDE (integrated development environment), you will find all of these associated in some way to the Controller.

```
[SerializeField] float m_MovingTurnSpeed = 360;
[SerializeField] float m_StationaryTurnSpeed = 180;
[SerializeField] float m_JumpPower = 12f;
[Range(1f, 4f)][SerializeField] float m_GravityMultiplier = 2f;
[SerializeField] float m_RunCycleLegOffset = 0.2f; //specific to the ch
[SerializeField] float m_MoveSpeedMultiplier = 1f;
[SerializeField] float m_AnimSpeedMultiplier = 1f;
[SerializeField] float m_GroundCheckDistance = 0.1f;
```

Each field completes a specific task.

Each “[SerializeField]” within the script is located within the Inspector View. This provides the ability to change components without the necessity of accessing the script itself. A very useful quality.

Go back to Unity and check. You will see something similar to this.

Most fields are self-explanatory. For instance Moving Turn Speed. Right now it is set to 360.
Run your game and have the character turn right.
Deselect Run. Change that Moving Turn Speed to 100. Run again.
See the difference?

Deselect the Run and reset the Moving Turn Speed back to 360.

The next feature is the Stationary Turn Speed. This is how quickly the play turns when stationary. Play with the numbers as you did before with the Moving Turn Speed and see what happens.

Next on the list is the Jump Power. This is what determines the force applied when jumping (and the jump height). If you plan on your character needing to jump on boxes, or leap over large objects, mess around with this number.

The Gravity Multiplier will determine how much gravity is involved when moving. Changing this to a higher number could cause your character not to jump very high. Having a high Jump Power and a low Gravity Multiplier can cause the character to jump off the island. Go ahead. Try it. Cool, huh?

The last few components are important factors. Let's quickly run through these.

- Run Cycle Leg Offset = animation cycle offset (0-1) used for determining correct leg to jump off
- Move Speed Multiplier = how much the move speed of the character will be multiplied by
- Animation Speed Multiplier = how much the animation of the character will be multiplied by

Turn this one up if you want a fast, cartoon run.

- Ground Check Distance = power of 'stick to ground' effect

Prevents bumping down slopes.

There is no right or wrong to these numbers. It all depends upon how you want your game to look, and feel to the user.

Go ahead and set-up your numbers on how YOU want YOUR GAME to be operated.

Remember to Save the Scene and the Project. Save Often.
https://youtu.be/eTJK7TkXwnY

Game Objects

You do not have to stop here with Ethan. If you have some models of Rocks, or other items to add to your island, go ahead and do so. Some ideas may be Signs, Boats, Shells, Sand Castles, and so much more. This is YOUR GAME. Make it exciting for you. Stay as simple as possible to keep file size low.

For this tutorial, we'll make a coin from a GameObject (Primitive), and then move this to form a prefab.

Let's get started.

Rez a Cylinder onto the scene (GameObject > 3D > Cylinder). Make the Size X=.5, Y=.05, and Z=.5.

Create a new material to add to the coin. Right Click within the Materials Folder and Create > Material. Name this, Mat_Coin. Within the Inspector window, find a color which resembles a coin to you. Add some "Metallic" to the color via the slider. Drag and drop the Mat_Coin onto the Cylinder. Rename the Cylinder to Coin.

Now, we are actually going to use this object in the game to be "picked up" by our player. This will be completed by adding a script to the Player. In order for this to work properly, a Tag will need to be created for the Coin. Before we do this, let's create an Empty Object to hold our coins (GameObject >Create Empty; Name this Pickups). This will just keep everything nice and organized. We'll be adding more coins later.

Drag the Coin into the Pickups to become "nested". Now here's an interesting piece of information, Prefabs CANNOT be nested into one another. This is why we created an empty object to "house" the coins.

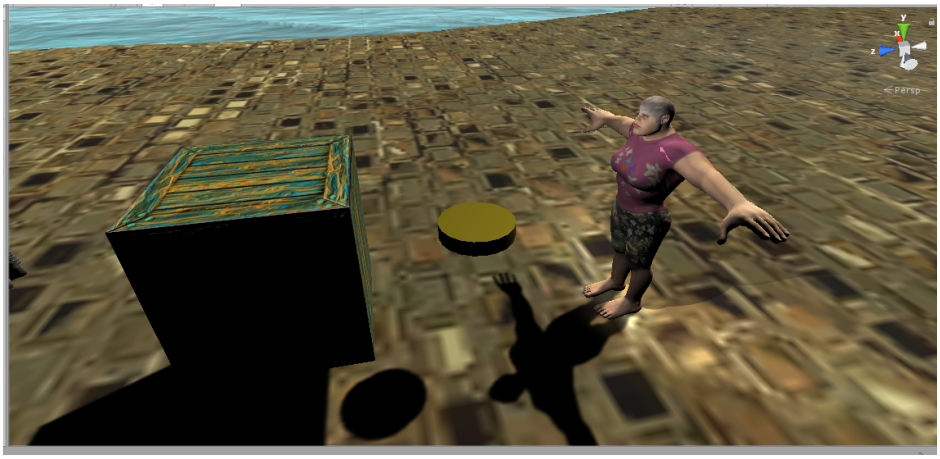
Select the Coin in the Hierarchy and move to the Inspector Window. Underneath the name of the coin, is a dropdown next to Tag. We will need to create a New Tag for the Coin. Select Add Tag. Name this New Tag "Pick-Up" without the quotes. This title is very important as it will be used in a script to call the object. Once the Tag is created, find the new Tag and tag your coin.

Let's continue making more Game Objects and make them Prefabs.

Prefabs

We'll make our coin a Prefab. A prefab is short for prefabricated. This is actually an "object" added to your assets. This GameObject stores its component values and allows for multiple copies to be added to the scene easily. These prefabs are called upon within scripts by Instantiating the object. Not only can copies be made quickly, editing the instance of the Prefab is simplified.

Create a folder named Prefabs (if you haven't done so) within the Assets if one is not created.



Now drag and drop that coin from the Hierarchy into the Prefabs folder. You just created a Prefab! How simple was that?! You can tell this is a Prefab because the title turned Blue within the Hierarchy. **Any Prefab is Blue**. Remember our Prefab Water? It is Blue in color for the title. Interesting…What else is a Prefab?

Now, create some more Prefabs for The Game. These can be as simple as a Crate to a Sign.
If you created a texture for a crate, just make a cube and add your texture.
If you would like to use some signs donated by Allan Jones, find them HERE.

https://youtu.be/zT7gSyyo4Fs

Collisions and Physics

Collisions and physics allow game objects to become “alive”. Making GameObjects react to a physical environment really makes the game come to life. This process is so simple to do in Unity. You will be amazed at how easy it is!

Let’s work with our coin to create physics so we can pick this object up.

Select the Coin in the Prefab Folder by double clicking. We must edit the Prefab for all edits to take on the coins within the scene.

In the Inspector Window, Add Component and select Sphere Collider. Check Is Trigger.
Next, select Add Component.
Type Rigidbody in Search.
Add select Rigidbody.
Within the Rigidbody Component, check Is Kinematic.
We will be adding some particles and a script to this object later. Normally, you would do all of these features at once as you are designing. Since we are so new, small steps are OK.

Save Scene/Save Project.

https://youtu.be/KqofmF_Ydec
NOTE: Not Shown in video...Add a Sphere Collider via the Add Component.
Make sure IsTriggered is checked.
YOU CAN DO THIS!

Particle System

Particles can make or break a game, just like sound in a movie (or game). These Components need to be thought out and used sparingly. Particles, if done too much, can cause Lag (slowing) of the game.

Let's add a "Bling" Particle to the Coin in The Game.
With the PREFAB Coin still selected, Add Component > Effects > Particle System.
Play around with the settings to get the effect you want.

Start somewhere and Run The Game. See what it looks like. You will probably wind up changing these numbers and colors often!

I cannot express enough on the use of the tools available to you for resources. Unity itself.
http://docs.unity3d.com/Manual/PartSysReference.html
Be sure to check out the Particle System Modules. This will help you make a better decision of what you are looking for in our effect.

Go forth and create effects!
Save OFTEN.

https://www.youtube.com/watch?v=iFimsUfX0YQ
NOTE: *Video from 2016. Still follows the same process*

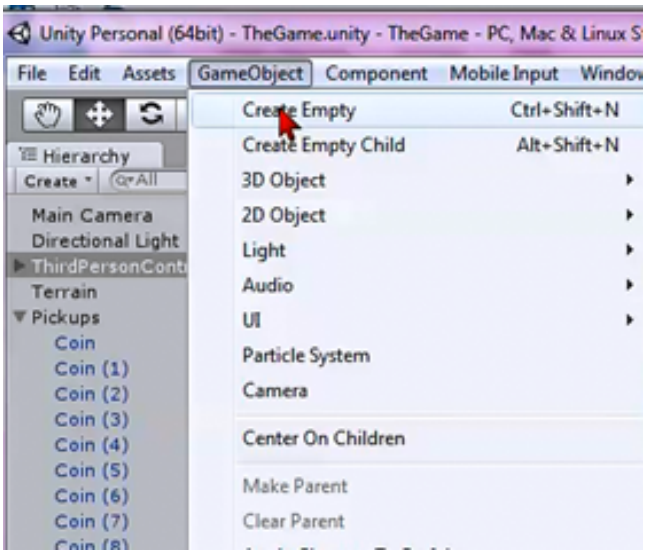
IMPORTANT!
If you have PINK SQUARES showing as a particle, the MATERIAL needs to be replaced. This is located within the renderer.

Particle “Poof”

You should have some type of particle effect with your Pick-Up object already created now. This effect is played as the object just “sits” in the game. What we’ll do now is add another particle effect to occur once the object is collected.

A way to think of this is to look at the effect of a waterfall. There is a mist (particle) as the waterfalls from the top point. Once the water hits the lowest point, the collision, there is a splash (particle). Make sense?

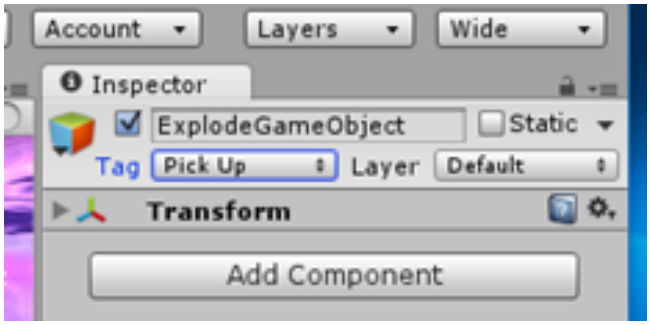
Let’s look at another example. A bullet is fired. The bullet has smoke (particle) following it. Once the bullet collides with the target, there is an explosion (particle).



To have this type of effect in your game, we need to make an explosion, or burst, or puff of smoke. Whatever it is that you wish to happen when the coin (or Pick-Up Object) is collected. This process will involve an Empty Object and a small amount of scripting. Don’t be afraid of that last part. We’ll make it as painless as possible.

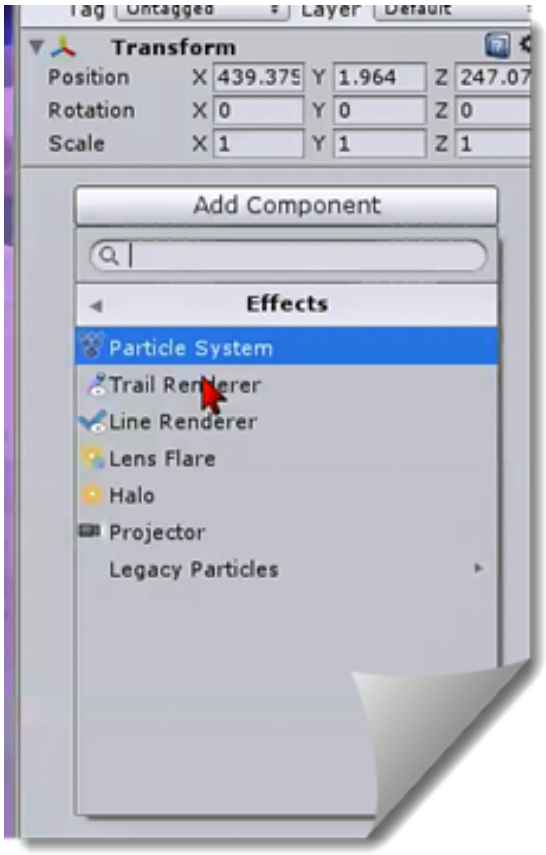
Create your Empty Object on the scene.

GameObject > Create Empty
Name the Empty Object. I named mine “Explode”.
Give this Empty Object the Tag of “Pick-Up”.

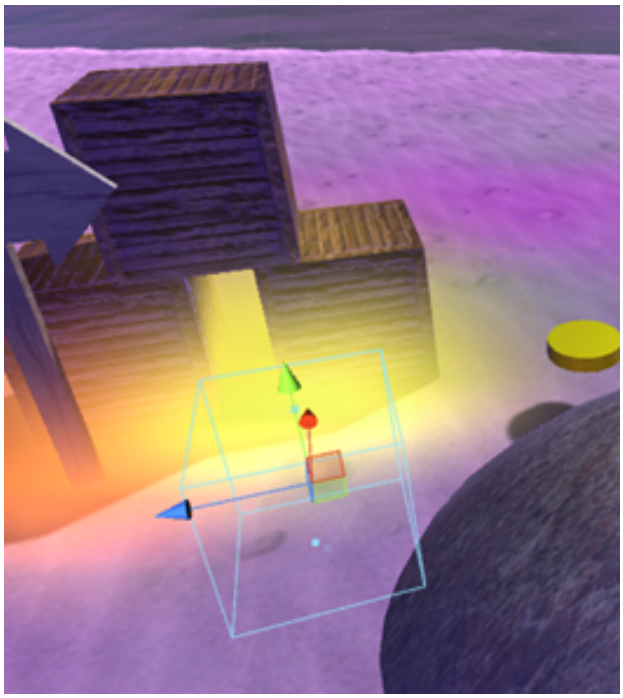


This part of adding the Tag allows us to call on this new effect with the Pick-Up Objects. Why make it harder by adding more objects and more scripts?

Add the Particle Effect Component to the Empty Object.

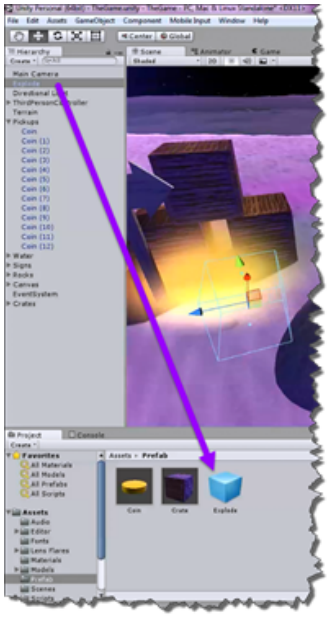


Play with the settings until you find what works best for your desired effect.

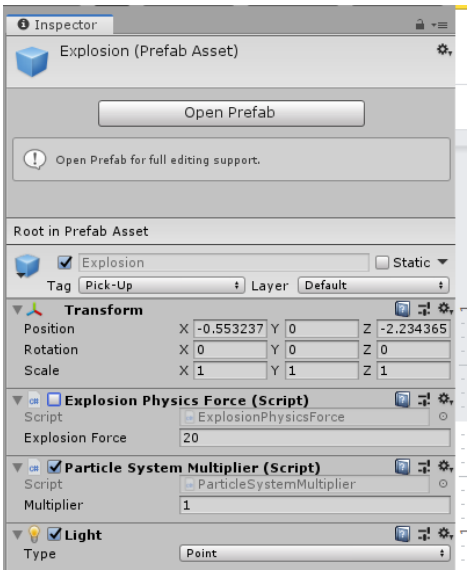


After this effect is at perfection, drag and drop the GameObject into the Prefab folder.
Delete the Object from the scene.

The reason we are making this particle a prefab is to be able to call upon it, or “instantiate” the prefab, as often as we need.



After all the creative design, we need to place on our logic thinking cap and work with Scripts in Unity. We will use the C# language, and the Visual Studio Integrated Development Environment (IDE).



This will come right after we bring in our Enemy, and set up the Navigation Mesh Agent, and Navigation Mesh area.

Now, if you are one of the ones to read first, and then go back and do, I have an easier process! Since we imported the Standard Assets, it is chalked full of particles! For my game, I am going to use the Explosion Prefab Particle. I selected this within the Standard Assets > Particle Systems > Prefabs folder. Here, I UNCHECKED the Explosion Physics Force, as I do not want the player to fly away after a pick-up. I then added the “Pick-Up” tag. That’s it.

Save your project, and let’s move on to the Enemy!

No Video for this one.

YOU CAN DO IT!

The Enemy

We have now arranged our game to hold an extra particle effect when collected, created Welcome, Win, and Lose Scenes, as well as kept the background music playing throughout. That's a lot of stuff!

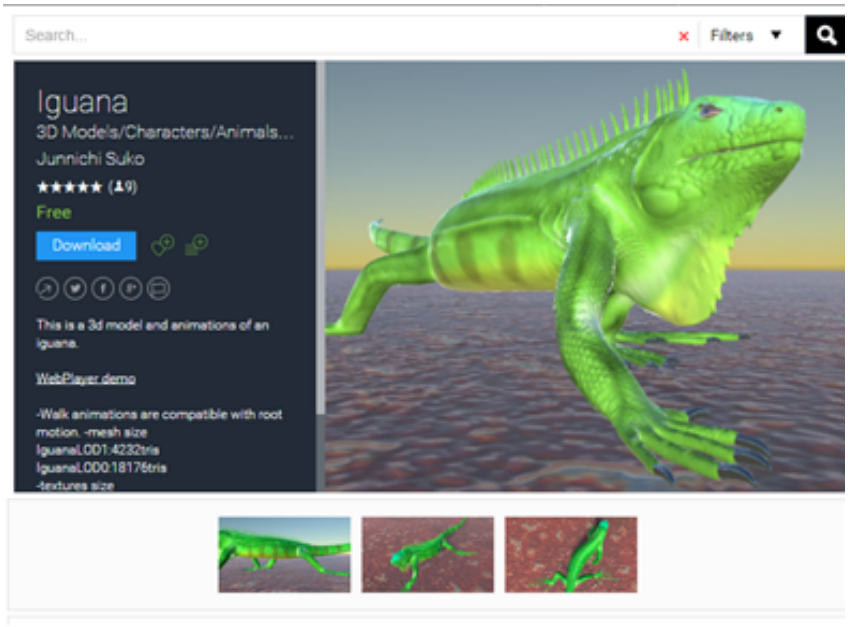
Now, The Game needs an enemy. It needs something to possibly make a player lose a game. This makes the game more of a challenge. Think about your game theme and what an enemy could be. These enemies could be large or small.

If you are talented, or want to try making your own enemy, take a look at the Maya Character Generator (https://charactergenerator.autodesk.com). You will need to create an account. This is FREE if you are using the Student Educational Account. Please know, there is a step involved in rigging this character. If you wish to try this, I will work with you, or you can use one of my characters.

You could use a day trial version of Fuse (https://www.mixamo.com/fuse/1.3/eol). This used to be FREE; however; Adobe purchased the product and now, it's not free, which is too bad. Please know, there is a step involved in rigging this character. If you wish to try this, I will work with you, or you can use one of my characters.

Best of all, you could search for something FREE on the Asset Store. This is what I am going to do for my game.

In fact, I found a FREE Iguana to call my enemy. This iguana package states it comes with the Animator Controller which is a big plus as I want this to interact with the ThirdPersonController. This way, we can create a "losing" situation.



For this part of the puzzle, we are just going to set our enemy up and have it operational. We'll worry about the controls in the next chapter.

The first thing to do is to open The Game Scene and import the enemy. Once the Asset Store import is completed, right click on the Asset Tab and select > close tab.

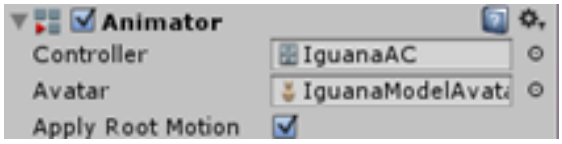
Next, drag your new enemy to the scene to test. Make sure it is located within the camera view. Select Run.

That's a pretty small iguana for an enemy, and it moves with the same keyboard controls as our player. I think I need to do some manipulation.

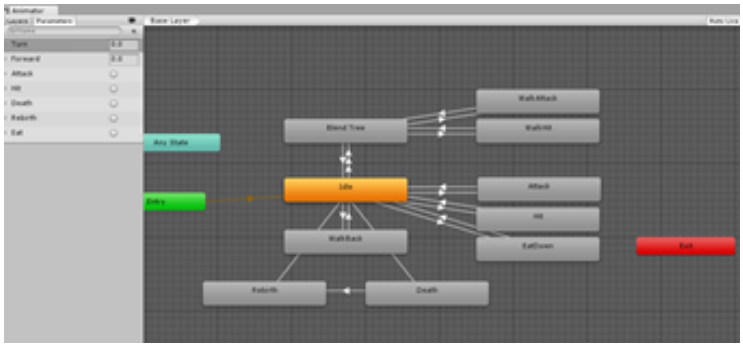
With the Iguana selected, change the Scale to X, Y, and Z = 5.
Select Apply after to set this with the Prefab.

That's a little better.

Take a look in the Inspector Window and notice the Iguana has its own Animator Controller.



Double click the Animator to open the Animator Window.
This is pretty complex.
It has the entry state which is idle and then, the whole system moves into different states, and blend trees.



Animator Controllers are very important and could be (and is) a class all within itself. For now, just know it is there and this is what the Iguana script calls upon to move the Iguana. In fact, open the two scripts associated with the Iguana in the Inspector Window.

In the Iguana UserController script, you can see what keyboard commands have been supplied to the Iguana. At the very bottom, the Horizontal and Vertical commands are Unity's way of saying the WASD and Arrow keys will move the Iguana.

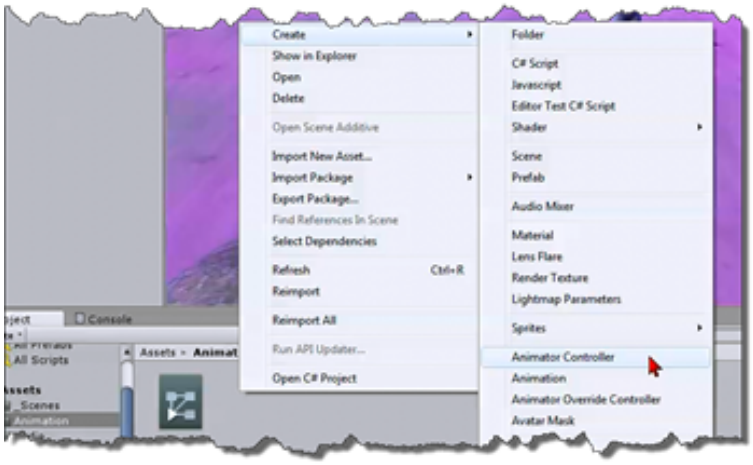
Interesting Stuff. For us, this is a Moot Point. We are going to create a Simple Animator Controller to move our "enemy". This controller will have two animations; a walk and an attack.

```
void Update () {
    if (Input.GetButtonDown ("Fire1")) {
        iguanaCharacter.Attack();
    }

    if (Input.GetKeyDown (KeyCode.H)) {
        iguanaCharacter.Hit();
    }

    if (Input.GetKeyDown (KeyCode.E)) {
        iguanaCharacter.Eat();
    }

    if (Input.GetKeyDown (KeyCode.K)) {
        iguanaCharacter.Death();
    }

    if (Input.GetKeyDown (KeyCode.R)) {
        iguanaCharacter.Rebirth();
    }



}

private void FixedUpdate()
{
    float h = Input.GetAxis ("Horizontal");
    float v = Input.GetAxis ("Vertical");
    iguanaCharacter.Move (v,h);
}
```

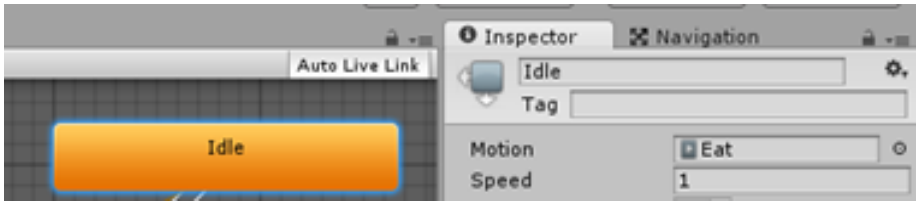
Animator Controller

Let's go ahead and create the new Animator Controller. We are doing this because we only need two animations for our game; a walk and an attack, or some would say, an idle.

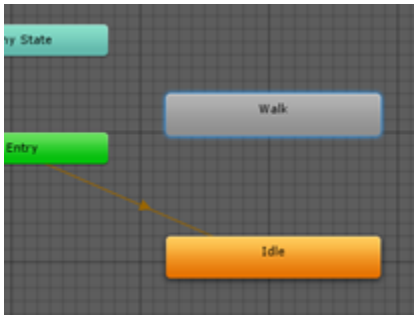
Within the Assets folder, create an Animation Folder if you did not create one earlier. Right Click within this folder and create a new Animator. Create > Animator.
Name this Iguana, or enemy.



Right Click *inside of the Animator Window* and Create > Empty. This creates a New State for the idle animation.
Select the New State and Rename this in the Inspector Window to Idle.
Within the Motion, find your Idle. I am using Eat.



Create another State, and name this one Walk. Add a Walk Animation. I am using Walk Attack.



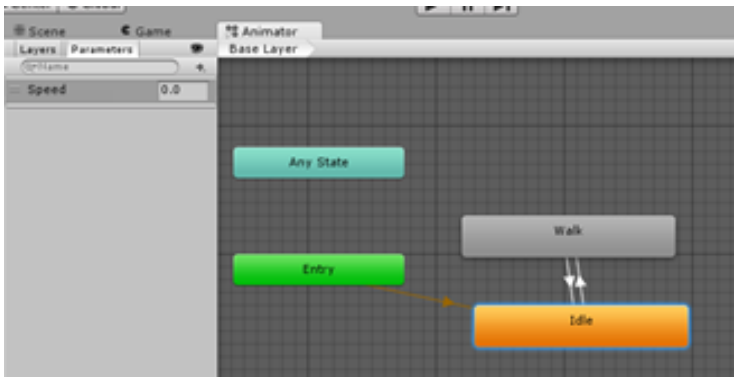
We need to associate the two States to Transition back and forth, plus add a float parameter for a script to call upon.

I know. Mind Blown. Just try to follow along here.

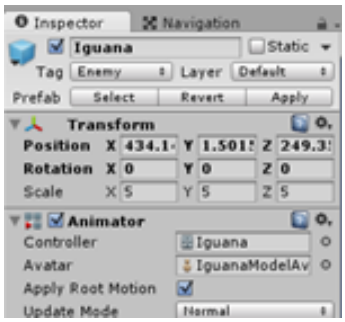
Right click on the Idle State > Make Transition. Drag the Mouse to the Walk and let go. Now do the same from Walk to Idle.

Make sure the Parameters option on the left upper side is highlighted and select the + sign to add a Float. Name this Float Speed.

You should have something which looks similar to this



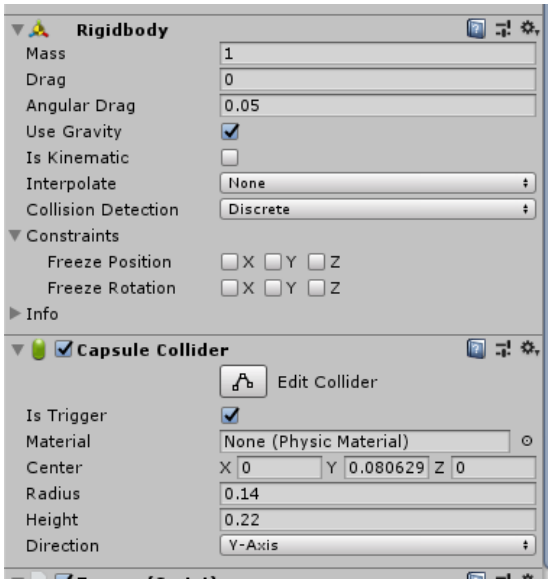
Now, select the Iguana in the Hierarchy Window and change the Animator Controller within the Inspector Window to reflect your new Controller.



We are now ready to make this baby come to life!
Save. SAVE. SAVE.
https://youtu.be/GGFRLnkVRvk
Please Note: The video shows the float name to be Float Speed. This should only be titled Speed.

Collider/Rigidbody

The Enemy that I am working with needs two more components to operate. It needs a Rigidbody, and a Collider. I'll be using the Capsule Collider for this, with Is Kinematic NOT selected. You may need to select the Edit Collider to make sure your capsule is large enough for the object. The Rigidbody needs to have IsTriggered selected. This will allow a script to call upon the Trigger to Operate a Method. We'll discuss that really soon!



https://youtu.be/xGedX8gspQs

NOTE: *Video shows me selecting the box for Is Kinematic...Don't select this!*

Nav Mesh

For The Game, our hope is to have the Enemy look at the Player and then advance toward the Player. If the Enemy Collides with the Player, the Game will be lost and the Lose Scene will Display. Sounds good? Let’s get started!

The process the Iguana follows is called Pathfinding, and it involves plotting the most efficient path to the Player. As the Player moves around, the Iguana's path has to change as well. Fortunately, Unity makes Pathfinding very easy.

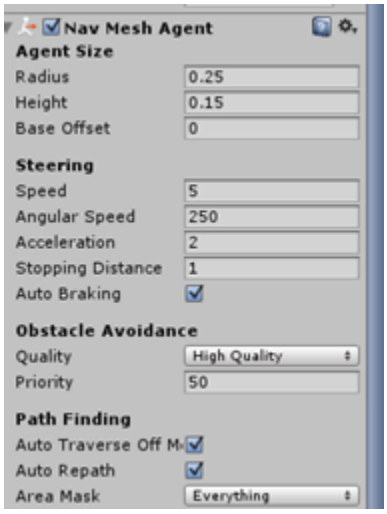
We use what's called a NavMesh to show the Iguana what parts of the playable area are navigable. Then, we set the Iguana to be what's called a NavMesh agent. Meaning that the Iguana has some basic artificial intelligence, and is able to use the NavMesh to walk towards a target intelligently.

Select the Iguana (Enemy) in the Scene.
Choose Add Component in the Inspector.
Navigation > Nav Mesh Agent.

This creates a Nav Mesh Agent component, with settings that control the physical size of the agent, how fast it can move, how fast it can turn.

Within the Inspector Window, set the radius to .25 and the height to .15.
These numbers may vary as your Enemy may be different. What you are trying to do is set the NavMesh size to be similar to that of the Game Object.

Set the speed to 5, and the Angular Speed to 250.
The Speed value is the maximum velocity, measured in Unity's units, and the Angular Speed is the maximum speed of rotation, in degrees per second. This value will allow the Iguana to turn quickly but still slowly enough to give the Player a brief opportunity to escape. Finally, set the stopping distance to five. This tells the Iguana to stop when it's close enough to the Player.



Click Apply to apply this to the Prefab IF asked.

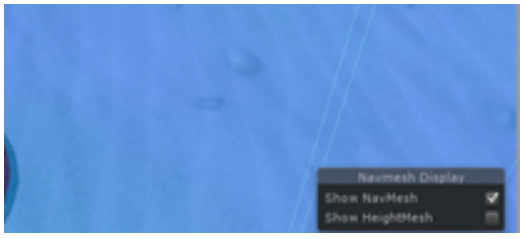
The next step is to create a NavMesh for the terrain and environment props so that any NavMesh agent in the game knows how to navigate the playable area.

Because I want all the environment to be used, I have created an Empty Object named "Environment" and moved everything related to the Environment inside of this object.
Do you remember how to do this?

Select the environment (Empty Object) in the hierarchy, and create a Nav Mesh, so that any Nav Mesh agent (the Enemy) in the game, knows how to navigate the playable area. Sound familiar? Do this by selecting Window > AI > Navigation.

At the top of the navigation window, click the Bake tab to modify the settings for the Nav Mesh. The NavMesh should share some of the settings of the NavMesh Agent we'll be using. That way, when the NavMesh does its calculations, it's able to determine if something like a rock, or ledge can be climbed by NavMesh agents with similar parameters. Sometimes you need to pick average values here for all NavMesh agents in a game. Because some NavMesh agents are taller than others, or have different dimensions and you have to match those to the NavMesh as best you can. We only have one Nav Mesh agent, the Enemy so we can use its exact values.

Within the Inspector Window, set the radius to .25 and the height to .15. Click Bake again.
Select Bake in the Bottom Right of the Inspector Window.
The NavMesh is highlighted in blue. It covers the playable area. This actually makes the scene very difficult for me to work in so I uncheck the navmesh checkbox to remove that blue.



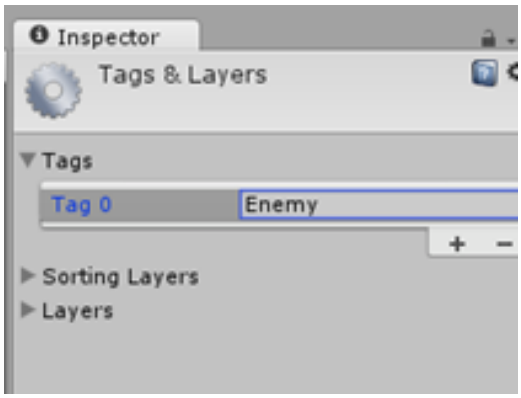
Save the Scene and the Project.
https://youtu.be/sxhrcqWDuGM
https://youtu.be/2rk7X3mDIWs (an extra...there are a few mistakes made here...can you see them?

Artificial Intelligence

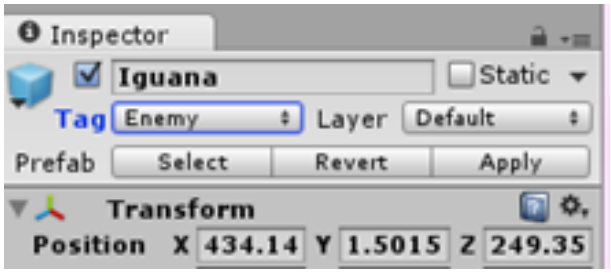
Artificial Intelligence (AI) in Unity is actually an Object which reacts to Triggers and Colliders. Sound familiar? Yes! You have created Artificial Intelligence with the Pick-Up objects.
It's that simple.

What we'll do now is create some AI with our Enemy. Just to make sure our Enemy is our "Enemy", check the Prefab for the Enemy tag. If it is not, here are the steps again.

Select the Iguana in the Scene Window.
Under the Inspector Window select Tag > Add Tag.
Select the Plus and Add Enemy.



Select the Iguana again and add the Enemy Tag.
Be sure to select Apply to set the changes to the Prefab if asked.



There are times when a Tag will never be used. It is better to have one than not; just in case.

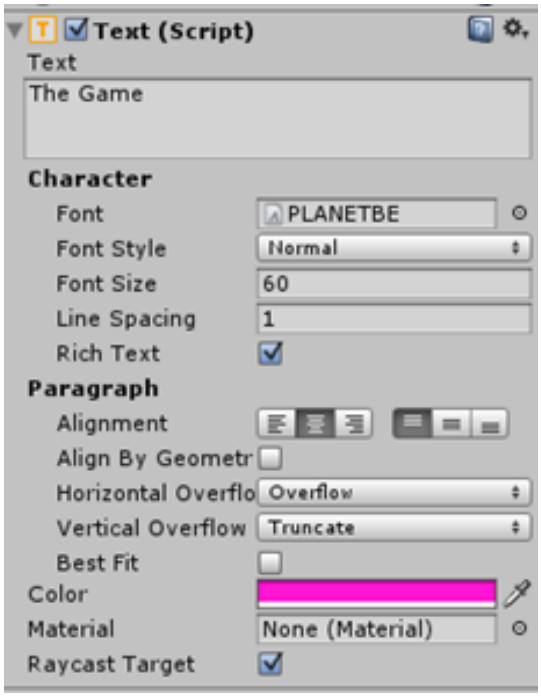
https://www.youtube.com/watch?v=5aPqdXs5sEc
NOTE: *Video Created in 2016. Still useful today ;-D*
Just to cover my behind, I am renaming the Enemy Game Object to "Enemy", and adding this to the Prefabs folder I created...This is to make sure the changes are solidified. Be sure to select "Original Prefab" if asked.

UI

We will add a UI (User Interface) to each scene with information and create “buttons” to advance our users to play the game or quit.

Open the Welcome Scene.
Change the Main Camera to display a Solid Color.

Select GameObject > UI > Text
Name the Text GameTitle
Add the Title of your game in the text area. Choose your colors and font. Align Center.
Set the Vertical and Horizontal Overflow, to Overflow. The image below shows the default Truncate within the Vertical Overflow.



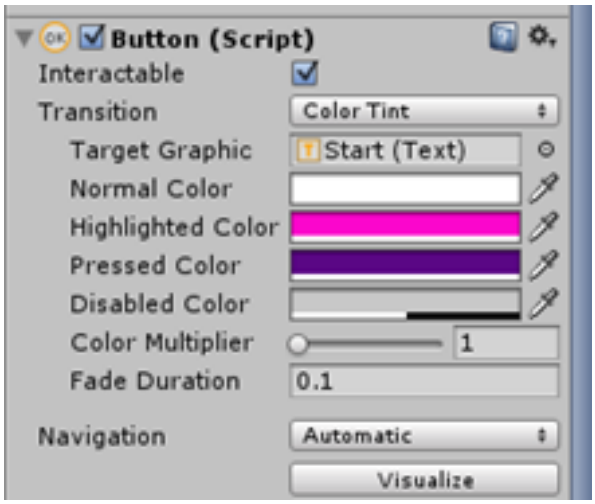
Duplicate the GameTitle text and rename it to Instructions.
We’ll need to reposition the two texts so they are not on top of each other.

Change the font to something readable and make the instructions brief and a smaller font size than the GameTitle. Change the color to something other than the Game Title. Make sure it won't get lost in the background color.

Duplicate the GameTitle text again and create a Start option. Move this into a position below the last two. Use White as the font color.

Got all that?

Select the Start text in the Hierarchy. Add a UI Component of Button within the Inspector View. Make sure the transition is set to Color Tint. This will allow us to change the color when the User hovers over the text and change the color when they click on the text.



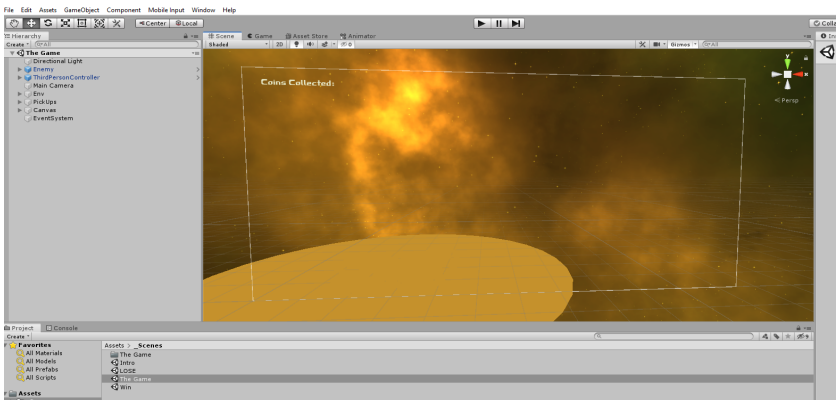
Change the Highlighted Color and the Pressed Color.
Be sure neither colors disappear in the background color. It happens…
TEST.
Yep.
Everything works together.

Now, do this same process for the Win and Lose Scenes. These should carry a message of Winning, or Losing with an option to RePlay or Quit. When we get to the Level Manager portion of the textbook, we'll be commanding the Replay to move back to the Welcome Scene.

One last little Text UI. This one will go on The Game Scene. Open this scene if it is not already open. Add your UI Text: Component > UI > Text

Rename this Count. Place it in the Upper Lefthand Corner of the screen. Be sure to use a text which is easily readable, as well as a color which won't get lost in your sky.

Save. Save. Save.



What I am going to do within the videos, is remove my scenes, and DUPLICATE the one which is set to go. I'll rename these Win and Lose, and make changes to the text information for each scene.

Because I am removing scenes which were part of a Build...remember that (File > Build Settings)?...I'll need to add these back into the Build after I am through creating the new scenes.

https://youtu.be/Yva9uMVsJlk
NOTE: This video shows me duplicating scenes. If you have your scenes created, you can delete the Win and Lose Scenes and follow along. This process allows all buttons to be the same.

Alos, don't forget about the build process. The Scenes need to be in order of Intro, Game, Win, Lose. The last two Scenes are not as important to orser as the first two.

Scripting

Now that we have our basic objects, let's start adding the C# scripts to make everything come to life! This class will not dive too deep into the actual programming, there's another class for that. We'll just introduce everything here.

You have already provided some scripting within The Game. Remember way back when you added the Smooth Follow to the Camera? That is a C# script!

Since this is a basic class, we will not get too heavy into scripting. We just need a few simple scripts for The Game to run correctly.

One script will be used to spin our coins. Another, we will add to the Player to control the Pick-Ups and Score. We'll also need an Enemy Script. Music Player, and Level Manager script. These last two will be looked at when we build those specific GameObjects.

Let's first take a look at some of the components to a script. According to Wikipedia (http://en.wikipedia.org/wiki/Programming_language), "a programming language is a formal constructed language designed to communicate instructions to a machine, particularly a computer."

This makes perfect sense. In Unity, the language is used to provide these instructions in order to make objects react to "things". These commands can also constitute keeping score of time.

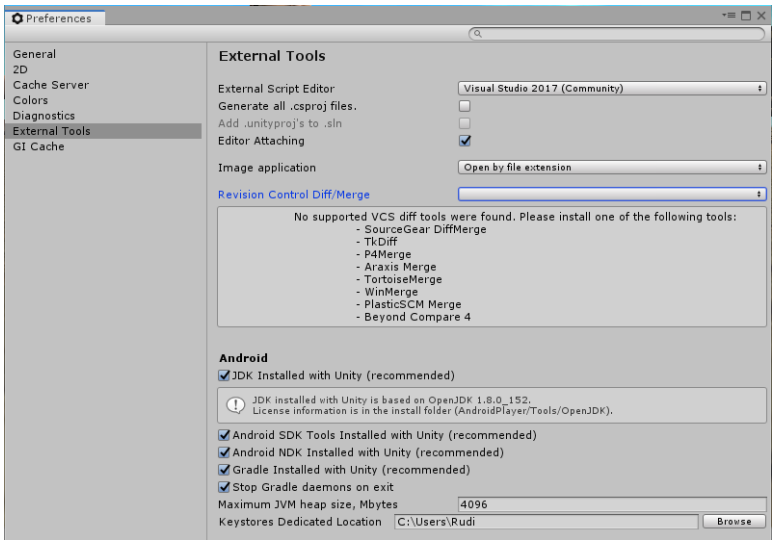
Unity had three languages it uses to control behaviors; C#, UnityScript, and Boo. Boo is no longer supported as of Unity version 4.6. For our class, we will dive into C#. Just understand that all languages hold the same "stuff". They are just written differently. We may even compare a few scripts inside of Unity.

With that said, let's create our first script. Create a new folder (if no Scripts Folder exists) and title this Scripts. Right Click within the folder and Create > C# Script. Name this Script Rotator. With the script selected in the folder, click Open in the Inspector Window to open the IDE (integrated development environment).

According to Wikipedia (http://en.wikipedia.org/wiki/Integrated_development_environment), this "is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools and a debugger. Most modern IDEs have intelligent code completion."

Unity uses Visual Studio for its IDE. You could write your scripts in anything you prefer. We'll just stick with the Unity Visual Studio for our purposes. If Visual Studio fails to open for you, the default needs to be set. Simple to do. Navigate to Edit > Preferences. Select External Tools. In the dropdown next to "External Script Editor", select Visual Studio 2017 (community). This will set Visual Studio to the default IDE editor.

You should see something like this



Now, we can start adding our scripts.

The Pick-Up

We are going to start with our coin. For this Game Object, we'll add the Rotator Script.

Open the Scripts folder, and right-click to make a new C# script. Name this Rotator. Double click to open the Visual Studio IDE.

The first two lines of the script are where we import any required namespaces, or libraries, which will be needed for the script to work.

The next line is our class "declaration", followed by opening and closing curly brackets {}. This is the script class. Everything within those curly brackets will be a part of this class; the Class Content. Everything we need will be placed within the {}.

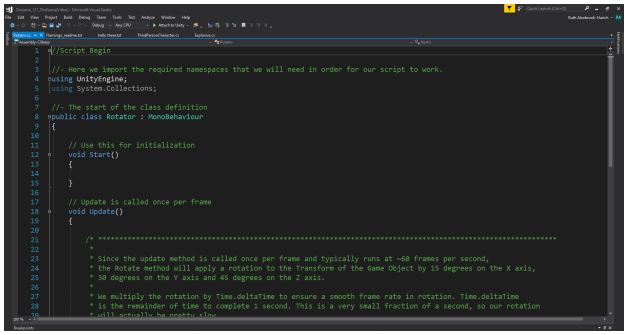
A script will also house Variables and Methods. For example, the void Update is a Method. Every 60 seconds the game, while in Run Mode, will update what is in this area. Within the void Start Method, is everything that will begin when the game is started.

Replace everything with the following from //Script Begin to //Script End.

```
//Script Begin

//- Here we import the required namespaces that we will need in order for our script to work.
using UnityEngine;
using System.Collections;

//- The start of the class definition
public class Rotator : MonoBehaviour {

	// Use this for initialization
	void Start () {

	}

	// Update is called once per frame
	void Update () {

		/*
*****************************************************************************************************************
		 *
		 * Since the update method is called once per frame and typically runs at ~60 frames per second,
		 * the Rotate method will apply a rotation to the Transform of the Game Object by 15 degrees on the X axis,
		 * 30 degrees on the Y axis and 45 degrees on the Z axis.
```

```
                *
                * We multiply the rotation by Time.deltaTime to ensure a smooth frame rate in
rotation. Time.deltaTime
                * is the remainder of time to complete 1 second. This is a very small fraction of a
second, so our rotation
                * will actually be pretty slow.
                *
                *
**********************************************************************************************************/

                transform.Rotate (new Vector3 (15, 30, 45) * Time.deltaTime);
        }
}
//Script End
```

You should now have something which looks similar to this



All words after the two forward slashes // are comments. Commenting in scripts is very important. It is something we should all practice. This provides us with details on what is doing what, what is being called where, and why we added what we did!

There is also an area with a large chunk of commenting within the middle of the script. Note the /* at the beginning and the */ at the end. This allows for large areas to be commented on.

Take a look at these comments and see if anything begins to make sense. When you are ready. We are going to add this script to our Coin Prefab.

FIRST, you need to save the script! You will know when the Visual Studio has not saved a script for there will be a circle in the scripts tab. Take a look. Then Save the Script.
This can be completed by File > Save or a simple Ctrl + S or Command + S.

Once the script is saved, navigate back to Unity and Open the Prefabs folder. Select the Coin. In the Inspector Window, Add Component > Scripts> Rotator.

Run The Game. The Coins should now spin on the X, Y, and Z Axis. How cool is that?!

https://youtu.be/n4oKbtUwgP4
NOTE:*In the video, I show another way to create a script. There's also a typo...do you see it? You can also find the scripts HERE*

The Player

We now must add a script to our Third Person. This script will allow the Coins to be Picked Up, and display the “score” and “win” screen. When we add this one, the game will come to life more than before!

Let’s add this script the same as with the Rotator. Within the Scripts folder, right click and create a C# script. Be sure to name this Player. Double click to open the script in Visual Studio. Replace EVERYTHING in your script with the following:

```
//Script Begin

//- Here we import the required namespaces that we will need in order for our script to work.
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
using UnityEngine.SceneManagement;
//- The start of the class definition
public class Player : MonoBehaviour {
	//- Private Variables
	//- Are specific to this class and cannot be manipulated through the Unity GUI
	private int count;
	//- Public Variables
	//- Can be accessed via the Inspector in the Unity GUI
	public Text countText;
	//Prefab Addition
	public GameObject explodePreFab; //placing the Prefab into memory
	private float time; //Provides a time for our Prefab to stop playing
	private GameObject instantiatedObj; //allows us to instantiate "call on" our Prefab

	// Use this for initialization
	void Start () {
		//- Here we just initialize a few of our game objects and variables.
		count = 0;
		SetCountText ();
	}
	// Update is called once per frame
	void Update () {
		if (Input.GetKey("escape"))
			Application.Quit();
	}
	/// <summary>
	/// Raises the trigger enter event.
	/// This event is called by Unity when an object collides with another that has its collider
	/// set to be a trigger.
```

```
        /// </summary>
        /// <param name="other">Other.</param>
        void OnTriggerEnter ( Collider other ) {
                //- Here we just compare the tag of the colliding object and check if it's Pick Up
                if (other.gameObject.CompareTag ("Pick-Up")) {
                        //- If it is, we deactivate the other game object
                        other.gameObject.SetActive(false);

                        //Set Explode
                        instantiatedObj = (GameObject) Instantiate(explodePreFab,
transform.position, transform.rotation); //Calls the Prefab. Brings it into the same location.
                        Destroy(instantiatedObj, 5f); //Stops the object after five seconds.
                        //- Then we increase our count variable by 1

                        //Finds Audio Source. Plays the audio File
                        AudioSource audio = GetComponent<AudioSource>();
                        audio.Play();

                        count = count + 1;
                        //- We then call the SetCountText method.
                        SetCountText ();
                }
        }
        /// <summary>
        /// Sets the count text.
        /// This method is responsible for updating the countText UI object.
        /// The method also checks to see if your count is greater than or equal to 9
        /// when it is, the winText is set appropriately
        /// </summary>
        void SetCountText(){
                //- Update the text property of the countText GameObject by stringing together
the string value of the count variable.
                countText.text = "Collected: " + count.ToString ();
                //- Check if count is greater than or equal to 13
                if (count >= 1) {
        Invoke("DelayedAction", 1f);
                }
        }
    void DelayedAction()
    {
        //- When it is, set the text property of the winText GameObject to a message.
        SceneManager.LoadScene("Win");
    }
}
```

//Script End

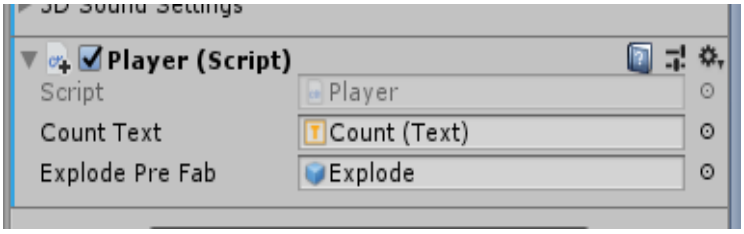
Save your Script.

Read the comments. There is a lot of “stuff” going on in this script! Note our Public Variables of the Count and Win Text. We will need to do something about this area once we add the script to the player. Let’s go ahead and do this.

With the Scripts folder open, select the Player Script and Drag this to the Third Person Controller in the Hierarchy Window. Now, select the ThirdPerson Controller. Look closely in the Inspector Window.

One more step is needed for this process to be completed. Something you have already done before. Think about the word “Public” in the script and what else was “Public” and what needed to be added to the Inspector Window. Is it coming back to you?

That’s right! We need to add the Explode Prefab to script within the Inspector View. This can be done by clicking the radio button and finding the Prefab; or, by just dragging and dropping this into the selection.



Save the script. Save the Scene. Save the Project. Run your game. Watch the magic happen!

Collect your single coin and make sure everything works!

Did you get a Red Warning in the Console about an Audio Source? If so, add this Component to the Third Person Controller. This is where you want to add your Pick-Up SOUND. Be sure to UNCHECK the Play on Awake. We do not want this sound when we start playing; only when we pick up our coin.

If needed, you can find the Scripts HERE
https://youtu.be/622ZiI3sqRg

Be sure to test your game after adding the scripts. The Player Script is written to only collect ONE coin. If the Collision is correct, and the Third Person Controller hits the coin, the score should reflect 1 and the win screen should appear.

The Enemy

Now we need to manipulate the Enemy to chase the Player around in the game. We'll do this by adding a script to your enemy.

Create a new C# script in any way you prefer. Remember, if you use the Add Component to the enemy itself, this will create the script in the main folder. You will need to move this to the Scripts Folder. File Management Folks!

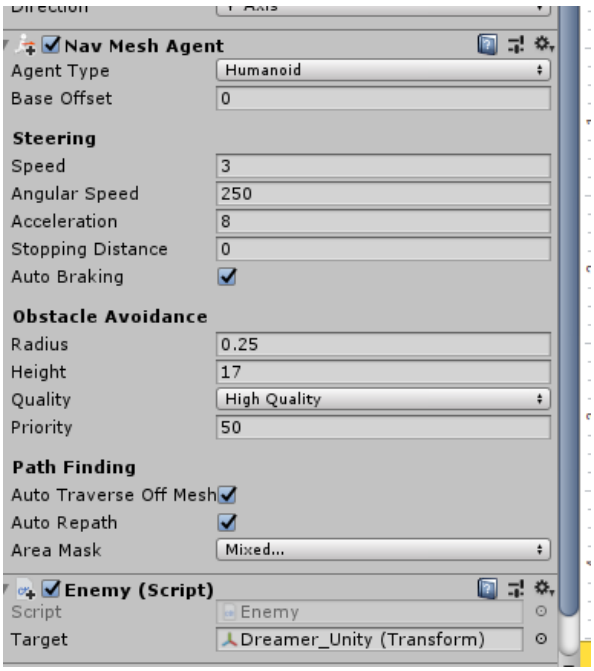
If you create a new script in the Scripts folder, this new Enemy script will need to be added to the Enemy, just as you did the Player Script.

Name this new Script Enemy.

```
//Script Begin

using UnityEngine;
using System.Collections;
using UnityEngine.SceneManagement;
Using UnityEngine.AI;

	public class Enemy : MonoBehaviour {

	[SerializeField]
	private Transform target;
	private NavMeshAgent enemyAgent;
	private Animator enemyAnimator;

	void Start () {
		enemyAgent = GetComponent<NavMeshAgent> ();
		enemyAnimator = GetComponent<Animator> ();
	}

	void Update () {
		//Set the enemy destination
		enemyAgent.SetDestination (target.position);
		//measure the magnitude of the navmeshagent velocity
		float speed = enemyAgent.velocity.magnitude;
		//pass the velocity to the animator component
		enemyAnimator.SetFloat ("Speed", speed);
	}

	void OnTriggerEnter ( Collider other ) {
		//- Here we just compare the tag of the colliding object and check if it's Pick Up
		if (other.gameObject.CompareTag ("Player")) {
```

```
                    //- If it is, we deactivate the other game object
                    other.gameObject.SetActive (false);
                    SceneManager.LoadScene ("Lose");
                }
            }
}
```

//Script End



Save the script and return to Unity. Add the Player target to the Inspector Window.

Run The Game without moving your character to see what happens. If your enemy seems to move too fast, use the Inspector Window to slow it down.

What we have done here is created an enemy to chase our player around the terrain as they pick up the objects. If the Enemy collides with our Player, The Game advances to the Lose Scene.

Take time out to read the comments on the script for a better understanding of what is happening in the game.

This script is very similar to the Player script in the Collision Method.
The difference relates to the Nav Mesh aspects we added.

After seeing the Enemy and Player in Run Mode, make any corrections to the Enemy Animator Controller if necessary. For myself, I see the animation I chose for the Flamingo does not look so good in play mode. I'll mess around with this design and see what happens. You may be lucky and all works perfectly!

If needed, you can find the Scripts HERE
https://youtu.be/eljkk1ZFhH4

Level Manager

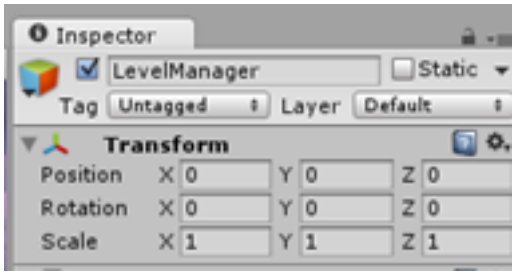
If you have tested, and you should have, you will notice the scenes advance when the coins are collected or the enemy attacks. Right now, we are unable to select the Replay or Quit options. We need a way to make this happen.

A Level Manager, or Scene Manager, will help in this process. It's a simple Empty Object and a Script.

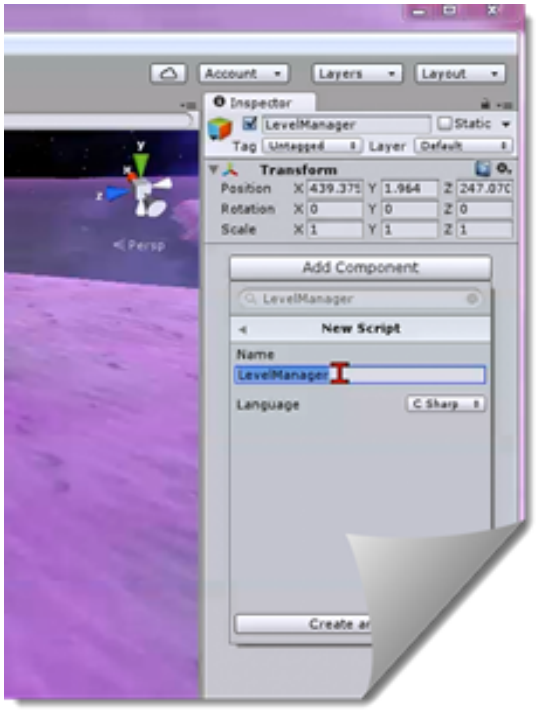
I am going to start with the Intro Scene Open.

Create an Empty Object and name this LevelManager.

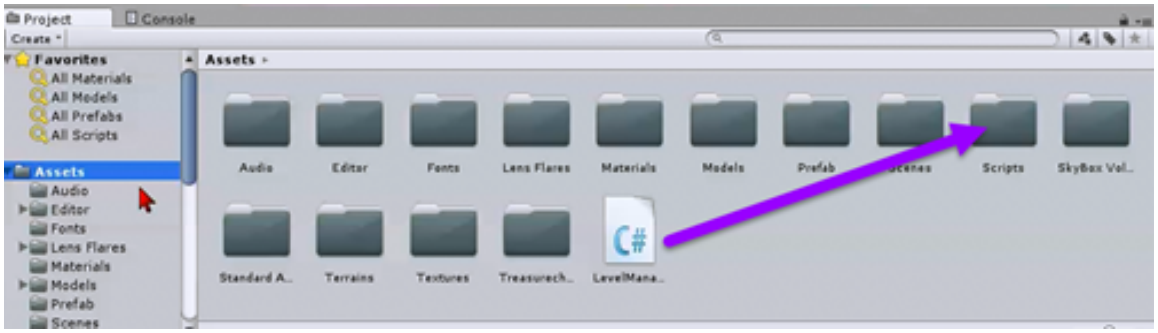
Reset the location of this Empty Object to X = 0, Y = 0, Z = 0. This action just moves the object to a central location. We do not want a bunch of stuff hanging around the scene.



With the LevelManager selected, create a C# script with the same name (levelManager).



Because this was created via the “Add Component”, you will need to access the Assets Folder and move the script into the Scripts folder. Just drag and drop.
Remember. File Management. File Management. File Management.



Open the levelManager script and replace the default with the following

```
//Script Begin

using UnityEngine;
using System.Collections;
using UnityEngine.SceneManagement;

public class levelManager : MonoBehaviour {

        //Makes the function Public so it can be called on at any time during the game
```

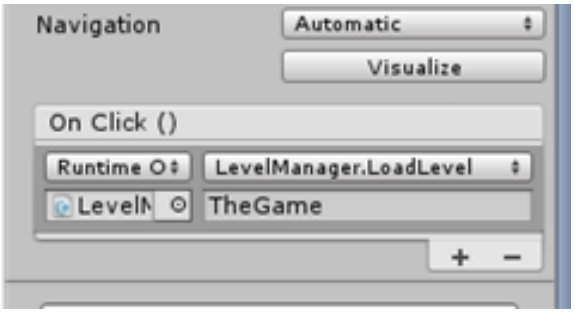
```
	public void LoadLevel(string name){
		Debug.Log ("Level load requested for: "+name);
		//this is a form of testing. The information within the quotations will display in the
console to show us if everything is working correctly.
		SceneManager.LoadScene(name); //States what Level will load
	}

	//Quit Request to exit the game.
	public void QuitRequest(){
		Debug.Log ("I want to QUIT!");
		//this is a form of testing. The information within the quotations will display in the
console to show us if everything is working correctly
		Application.Quit ();
		// Quits game. Does this mean we remove our original quit?
	}
}

//Script End
```

Take note of the comments and what is happening with this script. This is very important to understand.

We have added a “debugging” command. This Debug.Log is used frequently to verify that what we want to happen will really happen. It will display a read within the “Console View”.

Drag the LevelManager object from the Hierarchy into the Prefab Folder so we can call upon it later.

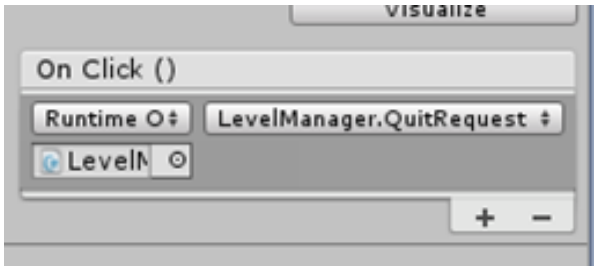
Open the introduction scene if it is not open. Select the “PLAY” Text in the Hierarchy. In the Inspector Window, within the Button Component, select the plus sign at “On Click ()”. Leave the RunTime Only as is. Right below this, add the Prefab “LevelManager” object. Under the Function, select LevelManager .LoadLevel. Add The Game level title to the empty space. SAVE!



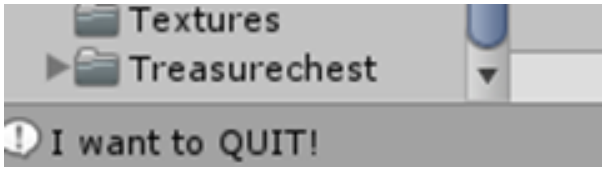
Run the game and select Start. If you start your game you are on your way! Excellent work.

Exit Run Mode and let's activate the Quit option.

Select Quit in the Hierarchy. Select the plus to add a command for the "On Click()". Add the LevelManager. Under Function, select LevelManager.QuitRequest. SAVE.



Test in Run. The game will not "close" in this way. Instead, look for the Debug.Log of "I want to QUIT!". This was our Debugging we added to the scripts. Great tool to use.
If you see this, all is well with the world.



All right. One scene down, two more to go. I want you to try these on your own.

Follow the same steps with the Win Scene and Lose Scene as completed with the Welcome Scene. You'll need a Replay and Quit option button commands. The Replay should navigate the player back to the Welcome Scene.

Go forth and create. SAVE and TEST.

If needed, you can find the Scripts HERE
https://youtu.be/GsNgYED6ZNI

Music Player

While testing, you probably noticed there is no music on any of the scenes. This is most regrettable. We need music, and environmental sounds in EVERY Scene! In fact, we imported sound waaayyyyyy back in the preparation of the project. Do you remember doing that?

To alleviate our conundrum, we are going to add a special Music Player to our project. Just as you created the LevelManager, a MusicPlayer needs to be created. I'll be adding mine to the Intro Scene. This MusicPlayer will then be added to the prefab folder and to each Scene Hierarchy.

If you added any music to the Main Camera, remove this. If you did not, don't worry about it. Remove the Audio **Source** from the Main Camera; NOT the Audio Listener.

Create an Empty Object.
GameObject > Create Empty.
Name this MusicPlayer.
Reset Position to ZERO.
Attach a C# Script with the same name; musicPlayer.
Be sure to move this script into the script folder to keep the nice File Management going!

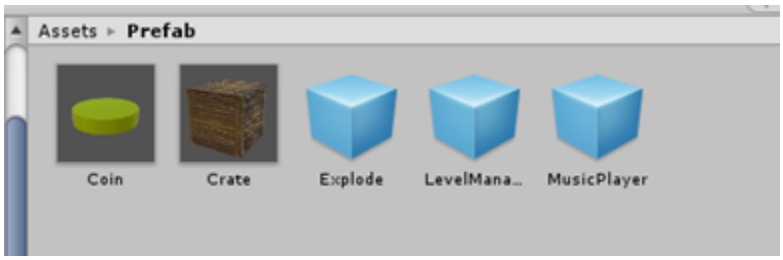
The Script should read as follows

```
//Script Begin

using UnityEngine;
using System.Collections;

public class musicPlayer : MonoBehaviour {
    static musicPlayer instance = null;

    void Awake () {
        if (instance !=null){
            Destroy (gameObject);
            print ("Self Destructing 2nd Music Player"); //stops the use of a second player to play
        }
        else{
            instance = this;
            GameObject.DontDestroyOnLoad(gameObject);
        }
    }
    // Update is called once per frame
    void Update () {
```

```
    }
}
```

//Script End

Attach the music to the MusicPlayer Object within the Inspector Window.
Be sure Loop and Play on Awake is selected in the Inspector Window.
Drag the MusicPlayer to the Prefab Folder.



Open your other scenes and add the Prefab to the Hierarchy Window.
Save.
Test.

If the music played in the Welcome Scene and then again in The Game scene, you should be feeling pretty good by now!
That's a lot of stuff to do just to switch between scenes.
Simple to do; just a lot to do.

If needed, you can find the Scripts HERE
https://youtu.be/leuqfnKVdFg

Audio Mixer

We have wonderful background music in our game. Wouldn't it be great to add some environmental sounds to this? Maybe some beach noise, and birds. Whatever else you think would make the environment feel more realistic.

For my game, I found a nice "beach" ambience earlier when searching for the coin sound. It is a beach with seagulls. It is about 3 Minutes long.

The following sites provide island/beach sounds. You may have a different terrain and will need to search for what works in your environment.

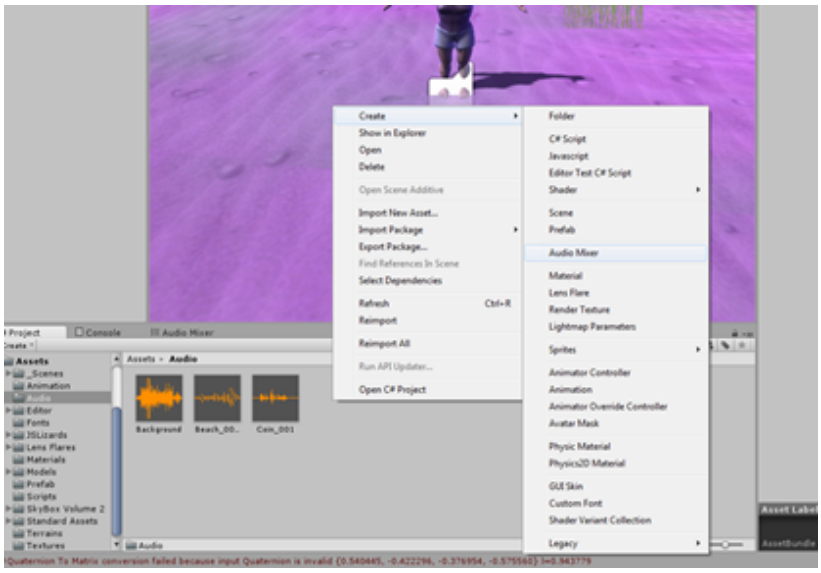
Please use copyright properties, and make sure the sound is a small file size. If needed, run it through the Audacity Program.

- http://soundbible.com/tags-beach.html
- http://soundbible.com/tags-island.html
- http://www.listeningearth.com/LE/general.php?pageID=8
- http://www.partnersinrhyme.com/soundfx/Ambience.shtml

Now, because I have chosen this one sound to play, I am just going to select the MusicPlayer in the Hierarchy, and add a second Audio Source to this object, adding the sound clip within the Inspector View.

Let's try it out in Run Mode.
It works; however, the background music is soft, while the environment sound is loud. Even the Pick-Up sound is loud. To fix this, we will need to create an Audio Mixer. To do this, right click in the Audio Folder and Create > Audio Mixer.

Name your Mixer something you can understand. Mine is known as “Sounds”.



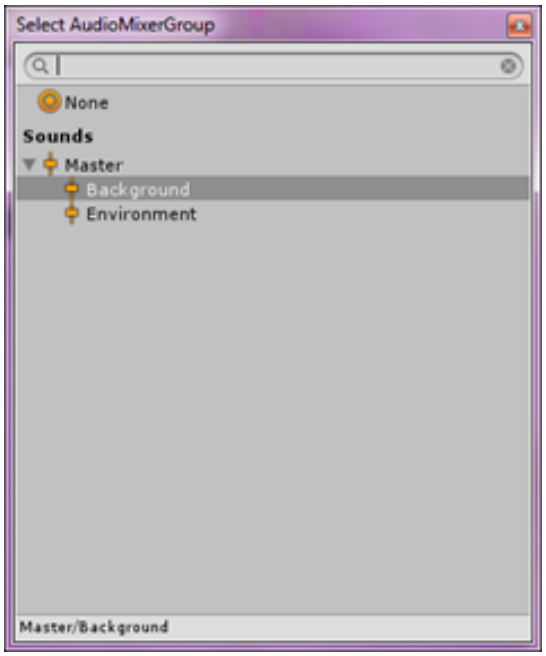
Double click the Audio Mixer to open it.

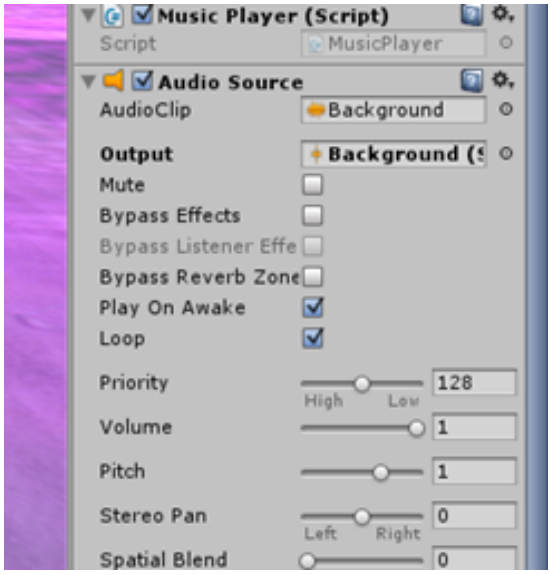
We are going to create a group so we can edit our sounds individually. Next to Groups, there is a plus (+) sign. Select this and create a group for Background. Add another for Environment. Add the Pick-Up if necessary. Be sure the main “Master” is selected when creating the second group.

Pretty Simple…

Now, we must associate the groups with the Music Player so we can edit their sounds individually.

Select the Music Player in the Hierarchy View and add the Background Group to the Background Audio Source in the Inspector View, and the Environment Group to the Environment Audio Source in the Inspector View.





With these sound clip sources associated, and the Audio Mixer View up, select Run. You should have the option to select the “Edit in Play Mode” at the top of the Mixer.

By selecting this option, you can adjust the volume of each audio group while the game is playing, and the changes will be saved.

Move the levels of the audio to your liking. Exit Run Mode and Save the Scene and Project.

Run again and listen to your sounds. You may feel that what you have just does not fit and you will need to make changes accordingly.

This is your game to do with as you deem necessary.

https://youtu.be/Zt-xv9lxmDI

Publishing The Game

We are now to the point where we can polish up our project, and publish so friends and family can play!

Number one to do will be adding the Pick-Ups to your island, and how you want your player to find these Pick-Ups. For instance, will you place your objects out and paint a path to follow? Will you direct your player with Signs? Will your player need to crouch or jump on a crate to collect any?

Whatever you choose, BE SURE to change Line 74 in the Player.cs to reflect the number you have chosen to use.

Wrap Up

You now have a game which has a character that runs about collecting objects. These objects have a sound and a particle which plays once they are picked up.

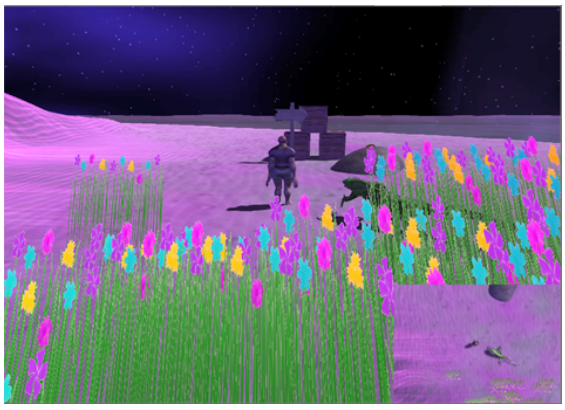
The Game also has an enemy which will attack the Character, thus causing the game to be lost. If there is a loss, or a win, new screens appear providing the player with an option to Replay or Quit.

There are also environmental sound effects as well as background music in The Game.

This is a lot of “stuff” and The Game is pretty awesome as is!

We could, however, add a few more things to make it even better.

One thing which could be done would be to add a footstep sound to our Third Person Character, or to your enemy. We could also play the game and find spots which the enemy seems to be stuck in and add more enemies to make the game more challenging. You could do this by placing the enemy in a spot, or by creating spawn points. The Game could also house a second camera to show an overhead of the game. This way, the player could spot the enemies easier. Let’s take a look at these one by one.



Footsteps (optional)

Creating the audio source is simple. You know how to do this. Find a Footstep sound and add it to the Audio Folder. Select the Third Person Character and Create > Empty Child Object. Rename this to Footsteps. Add an Audio Source Component to the Empty Footsteps object (Add Component > Audio > Audio Source). Add the Audio and uncheck “Play On Awake”.

You will want the footsteps to start and stop as the character moves about. That will be the tricky “coding” part. We will make a script for our Footsteps Object.

Add a Script Component to this object (Add Component > New Script) and name this footStep. Since adding this component via the Inspector Window, it will sit in the Root of the Assets. *You will need to move this script into your scripts folder; keeping everything nice and neat.* Replace the script with the following:

```
//Script Begin

using UnityEngine;
using System.Collections;

public class footStep : MonoBehaviour {
    public float stepRate = 0.5f;
    public float stepCoolDown;
    public AudioClip feet;

    // Update is called once per frame
    void Update () {
        stepCoolDown -= Time.deltaTime;
        if ((Input.GetAxis("Horizontal") != 0f || Input.GetAxis("Vertical") != 0f) && stepCoolDown <
0f){
            GetComponent<AudioSource>().pitch = 1f + Random.Range (-0.2f, 0.2f);
            GetComponent<AudioSource>().PlayOneShot (feet, 0.9f);
            stepCoolDown = stepRate;
        }
    }
}

//Script End
```

Save the Script. Add the audio to the script component within the Inspector window. Save the Scene. Save the Project. Test Run. If the footsteps are too loud, add a group to the Audio Mixer and lower the volume. If they are too soft, do the same; only raise the volume. Pretty Simple.

You can do this with your enemy as well. Just follow the same steps; so to speak.

https://www.youtube.com/watch?v=dQvaHpUGGdI

NOTE: T*his is an older video from 2016. The information is still the same.*
You will see a Different IDE used here, as well as a different Script CLASS. Be sure to use the above information for your script. You will also hear of an opportunity for EXTRA CREDIT!

The Enemy Again (Optional)

In my game, I noticed a few areas where The Enemy becomes stuck. I have two choices here. I could smooth the terrain and allow The Enemy easier access, or I could add more enemies throughout the landscape to make The Game a little more difficult.

What I'll do is add a Spawner to spawn enemies. In fact, I'll add a script to allow me to add different kinds of enemies in the future. Maybe I want more than one "attacker"! This allows me to be a little more creative.

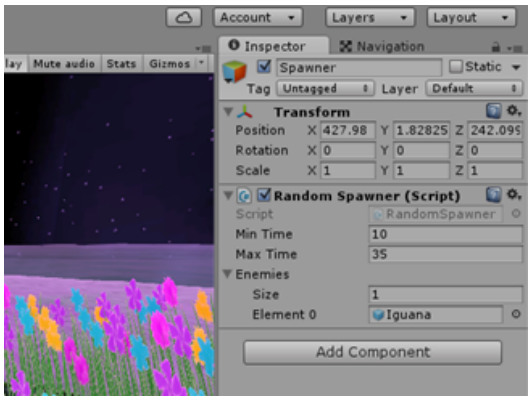
I'll try this first in the beginning area (near the original Enemy) and see what happens. To accomplish this, I'll create an empty object near the starting point.
Next, I'll add a RandomSpawner.cs to the object.

```
//Script Begin

using UnityEngine;
using System.Collections;

public class RandomSpawner : MonoBehaviour
{
   bool isSpawning = false;
   public float minTime = 10.0f;
   public float maxTime = 35.0f;
   public GameObject[] enemies;  // Array of enemy prefabs.

   IEnumerator SpawnObject(int index, float seconds)
   {
      Debug.Log ("Waiting for " + seconds + " seconds");

      yield return new WaitForSeconds(seconds);
      Instantiate(enemies[index], transform.position, transform.rotation);

      //We've spawned, so now we could start another spawn
      isSpawning = false;
   }

   void Update ()
   {
      //We only want to spawn one at a time, so make sure we're not already making that call
      if(! isSpawning)
      {
         isSpawning = true; //Yep, we're going to spawn
         int enemyIndex = Random.Range(0, enemies.Length);
         StartCoroutine(SpawnObject(enemyIndex, Random.Range(minTime, maxTime)));
```

```
    }
  }
}
```

//Script End

Once the script has been saved, the Enemies need to be accounted for. For now, since I only have one, I'll make this number 1. As I add more, I can change this number and add the associated game object to the array/list. Pretty cool huh?



Save the Scene. Save the Project. Test the game. If you find "dead zones" for the enemy now, maybe try duplicating the Spawner. Don't make the game too difficult or people will just not play it. Make it fun, and challenging.

https://www.youtube.com/watch?v=gDL6sHTQxFc
NOTE: T*his is an older video from 2016. The information is still the same.*
You will see a Different IDE used here.
*I speak of obsolete "errors". **These ARE NOT errors, they are warnings**.*

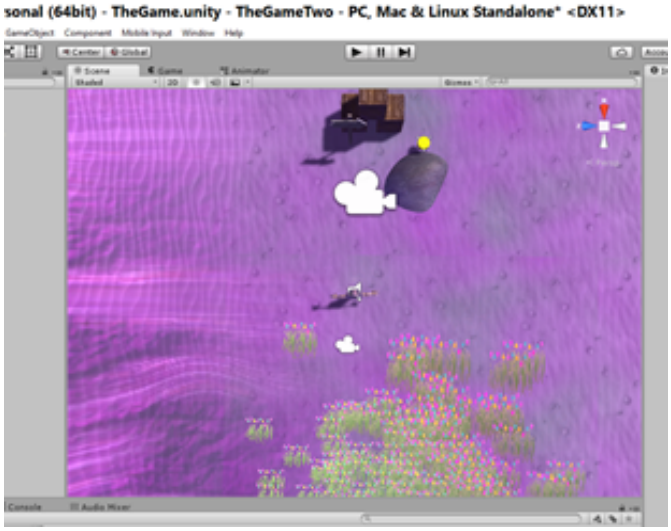
Second Camera

Now we need to add that second camera since we have so many enemies running around!

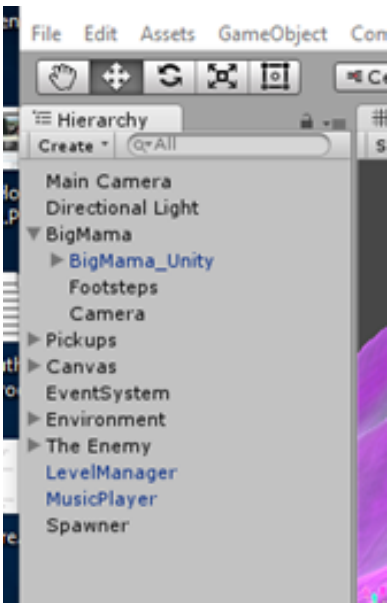
We'll use a camera to look down upon our player. This will allow us to see if any Iguanas are sneaking up on us.

Create your second Camera. Game Object > Camera

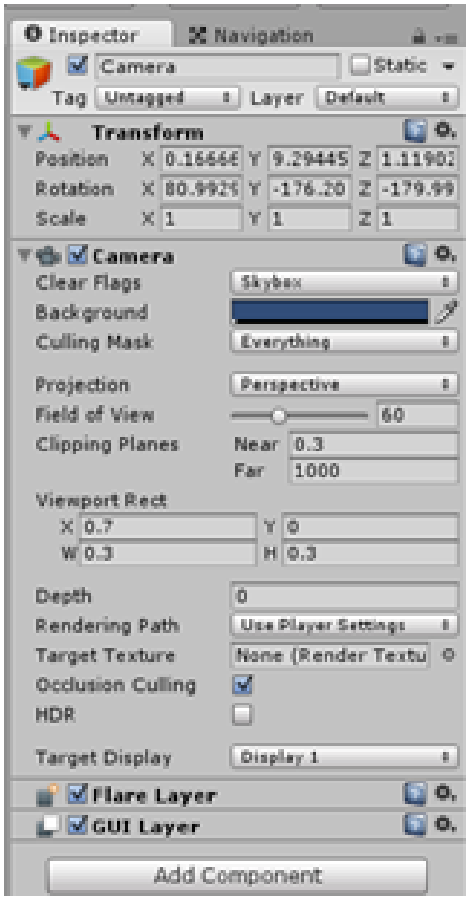
We are going to set the camera HUD to display in the lower right corner. First we should align our view from above. Move the scene to above the Avatar and then select the newly created camera > GameObject > Align with View.



Move the Camera to nest inside the Third Person Character.



Within the Inspector Window, set the coordinates of the Camera Viewport Rect as
X = 0.7, Y = 0, W = 0.3, H=0.3



These numbers actually locate the coordinates of the screen when a person plays the game. Just think in terms of the X and Y, with 0 being the bottom left and 1 as the upper right.

Now, just play the game a few times, see if this newly created HUD helps you to detect the enemies.

Speaking of enemies, it might be good to add a line in the Welcome scene.
Remember how to do this?

https://www.youtube.com/watch?v=p3FY8e9IIw8
NOTE: T*his is an older video from 2016. The information is still the same.*
You will see a Different IDE used here.
You will hear of another opportunity for EXTRA CREDIT.

Polish and Deploy

We will now polish up our project and create builds for others to play our game. We will also create Packages from the scenes in order for me to review your work.

Let’s get started!

Polish

If you have been following along with the steps here, you should have a scene with an Island, a Main Camera with Audio, Directional Light, Water, a Third Person Controller, a Coin, possibly some Signs, maybe a rock, and a crate. I highly encourage you to add crates as they can be used to jump on or hide a coin between boxes.

What needs to happen to “Polish” this game is to first create copies of the Coin to hide around the Island. This is very simple to do. Select the Coin within the Hierarchy Window. Using the Keyboard Shortcut of Ctrl + D, copy the coin with as many copies as you want to be collected. For example, I wanted 13 coins hidden, so I made 12 copies of the coin. These copies are all in the same X, Y, and Z Axis.

Now, the coins need to be hidden around the island. Some people like to drag the coin around and place it strategically. For myself, it makes sense to first provide a directional sign for the player on which way to move. I will next, Pan and Zoom in that direction.

Once I find a spot to hide the Coin, I select the coin in the Hierarchy Window, then use the GameObject > Move to View. This moves the Coin into the location view, which can then be moved into the precise location. Once placed, Move to view the directional sign for the next obstacle. This moving of Objects sounds so simple, however, it took me a long time to figure out. Once I found its use, I have been a huge proponent of showing it to everyone!

Go forth and hide your coins. Save the Scene. Save the Project.

Because more Coins were added to the Pick-Ups, we need to change a number within the Player script. If this is not completed, The Game will be won when only one Coin is collected! Find the two lines in the Player Script which read

```
//- Check if count is greater than or equal to 1
                if (count >= 1) {
```

The first line is the comment. You probably want to change this to remember what is happening in the code. Change the number “1” in the Comment and in the line below, to whatever the amount of coins hidden in The Game. For example, I have 13 Coins hidden. The script should read

```
//- Check if count is greater than or equal to 13
                if (count >= 13) {
```

Save the Script. Save the Scene. Save the Project. Run The Game.

https://youtu.be/TjbLsd7ZtII

NOTE: *In the video, for time constraints, I only add two more coins. Add more in your project. TheThird Person animations do not show in Project Mode. These will play in the build!*

Deploy

Now that everything is ready in The Game, we need to “Deploy” it so our family and friends can play. This process is very simple to complete.

Go to File > Build Settings.
If you failed to add the Scenes, add them now, in order of play. This process should have been completed earlier in the Project
Select the Player Settings.

In the Inspector Window, you can change icons to customize what users see when they access the .EXE file. If you wish to use these, an image must be inside the project file to be selected.
Select Build and Run.

Unity will pop up a Dialogue Box asking where you want to save this Build. What works, is to create a folder by the games title (The Game), and save the Build files within this folder. This way, ALL components of the Build are within ONE FOLDER. Very important.

Now, sit back and watch the magic happen.
Unity will create an Executable File (.EXE), along with a Data Folder inside of the newly created The Game Folder. When sharing your work, ALL OF THESE FILES AND FOLDERS, within the newly created The Game Folder, must be shared.

The folder can be compressed and sent to friends and family via email. The files could be uploaded to a cloud storage for family and friends to download. So many ways to get that game to others to play!

Follow the same process for a WebGL Build. The WebGL will not play unless it is housed on a website. I have a website which I place games on for play. You can see these student generated games at https://www.yc.edu/v6/schools/bucs/video-game-development.html

If you are serious about displaying your projects for future providers, I’d suggest getting a website, and loading your work on this. It is not mandatory for this class. This is something you would need to invest time, energy, and money in. It is something to think about for the future.

How does it feel to play your first game?!
https://youtu.be/AvqP4apqndc

The Project Files

In the past, I used to have projects compressed and sent to me to review. This fails more times than it works these days, so we are going to try something different.

Open the scenes folder. Right click on the Intro scene and select “Export as Package”. When the dialogue box opens, create a NEW Folder to save the Exported Packages with.

Export EACH SCENE to this folder. We will then compress that folder to use. Be sure your name is on the folder.

We will then create a compressed file with the builds, and the packages to be submitted via the Learning Management Software (LMS).
https://youtu.be/KKeM1CD_pPc

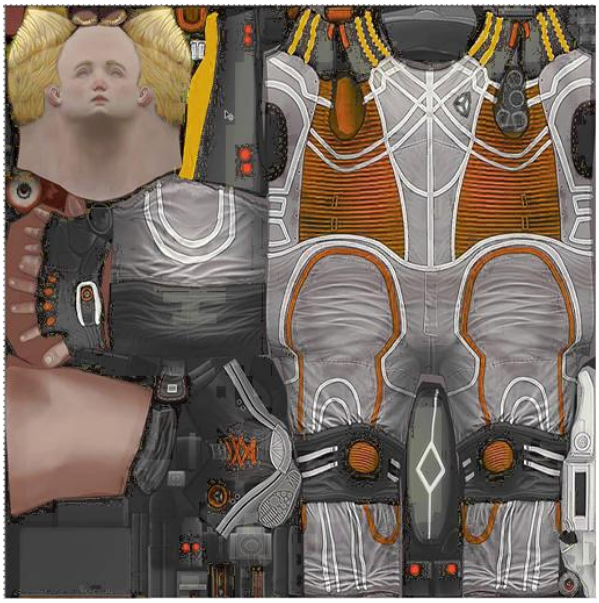
Summary

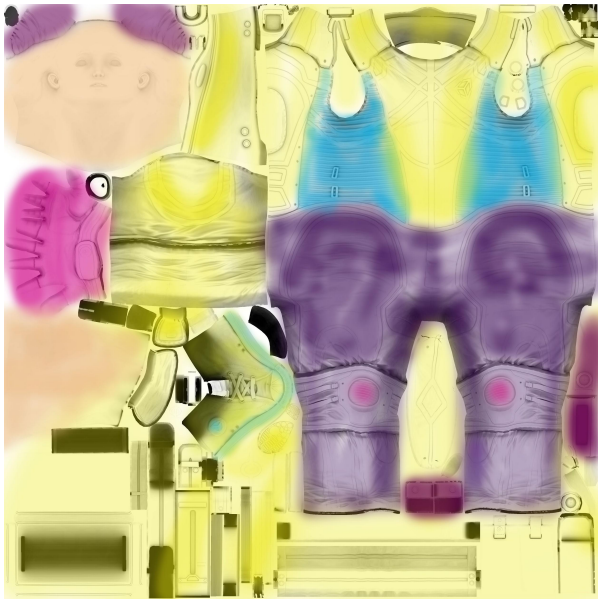
We have now learned the Basics of creating a Game with Unity. Some important features were shared here on File Management, the Unity Interface, Game Objects, Scripting, and so much more. It is just a “Start” to get acquainted with the Unity Application and all the other features available to you for FREE.

When using FREE stuff within a project, review the Creative Commons. Better yet, try to create your own art, sound, and models. This makes The Game YOURS.

You can make sounds, or crop music for loops by using a FREE Audio editor. Audacity is a great tool for this.

You can take plain, old, Ethan (Third Person), and work with his “texture” to change colors. For example, here is the original and altered texture for the material of Ethan. The latter is from a previous student’s work. Need the Ethan texture? Check the Announcements!





Using Adobe Fuse, or Autodesk Character Generator (free with a Education Account) to make your own third person is another great idea to make the game yours. Here is a quick video on how to change the “Rigged” Third Person in the game (https://youtu.be/CCDTIFa1Ih4). If you would like to try this route, I have many avatars you can use. Send me a request, and I’ll shoot one your way!

One of the most important things to remember is to have fun. Learn from mistakes. Growth cannot be achieved without failure. Learn to laugh at life…and be creative with your imagination.

Works Cited

Wikipedia. Unity. 14 May 20120. <https://en.wikipedia.org/wiki/Unity_(game_engine)>.
TeamTreeHouse 6 November 2018. <https://teamtreehouse.com/home>
Unity User Manual 14 May 2020
<https://docs.unity3d.com/2020.1/Documentation/Manual/UnityManual.html>

Assets

Please Note: *The list of Assets includes all past and present uses of assets. I keep these listed as some of the older videos are still used. I believe credit should always be provided to all persons when using their work.*

Cog Flower Texture by Ruth Alsobrook-Hurich
Daisies WhiteBlue Texture by Ruth Alsobrook-Hurich
Flowers Alpha Texture by Ruth Alsobrook-Hurich
GoldGreenWood Texture by Ruth Alsobrook-Hurich
Rust Paint Texture by Ruth Alsobrook-Hurich
Rust 2 Texture by Ruth Alsobrook-Hurich
Particle Test Texture by Ruth Alsobrook-Hurich
Dotted Line Rose 9 Texture by Xzavia Yifu of Timeless Textures in Second Life*
Jeweled Mermaid Gold 9 Texture by Xzavia Yifu of Timeless Textures in Second Life*
Jeweled Mermaid Gold 13 Texture by Xzavia Yifu of Timeless Textures in Second Life*
Stone's Throw Frozen 9 Texture by Xzavia Yifu of Timeless Textures in Second Life*
Tech Savvy Dark 9 Texture by Xzavia Yifu of Timeless Textures in Second Life*
Standard Assets by Unity (Asset Store)
Flamingo Assets by WDallgraphics (Asset Store)
Japanese Coin Assets by Gnarly Potato (Asset Store)
beachAmbience Sound by SFX - Storyblocks
Dreaming Sound by SFX - Storyblocks
Cheerful Days Sound by SFX - Storyblocks
Footsteps Sound by SFX - Storyblocks
Magic Turning Spell Sound by SFX - Storyblocks
Deadpack Font from DaFont
Dreaming Font from DaFont
Fairy Tale Font from DaFont
Fishes Friends Font from DaFont
Hidalgo Font from DaFont
Sharkbit Font from DaFont
Zombieland Font from DaFont
8K Skybox Pack Free by BG Studio (Asset Store)
* *Used with permission*